

les Cahiers du **Programmeur**

XUL

Mozilla XPFE, XPCOM, XBL, XPI, CSS, JavaScript, XML, RDF, DOM, PHP 5

Jonathan Protzenko

Avec la contribution
de Benoît **Picaud**

Préface de Stéphane **Mariel**



EYROLLES

AVAXHOMEDOTWS

Jonathan Protzenko

les Cahiers
du **Programmeur**
XUL

Préface de Stéphane Mariel

Avec la contribution de Benoît **Picaud**,
de Stéphane **Mariel** et de Jean-Marie **Thomas**
avaxhome.com

EYROLLES

www.frenchpdf.com

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Remerciements à Julie Meyer.



Le code de la propriété intellectuelle du 1^{er} juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée notamment dans les établissements d'enseignement, provoquant une baisse brutale des achats de livres, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans autorisation de l'éditeur ou du Centre Français d'Exploitation du Droit de Copie, 20,

rue des Grands-Augustins, 75006 Paris.

© Groupe Eyrolles, 2005, ISBN : 2-212-11675-6

www.frenchpdf.com

Préface

Ces dernières années, l'Open Source a démontré sa capacité à conquérir des pans entiers de l'informatique d'entreprise, au point parfois de devenir la référence détrônant les solutions établies. C'est le cas des services d'infrastructure, comme la messagerie avec les logiciels libres Sendmail ou Postfix, ou la publication web avec Apache.

Toutefois, il n'en va pas encore de même pour les applications présentes sur le poste de travail de l'utilisateur. Le plus souvent les logiciels Open Source sont perçus comme des suiveurs, se contentant de copier des fonctionnalités déjà disponibles par ailleurs. D'où un sentiment de décalage permanent.

Cette vision est cependant dépassée et ce cahier du programmeur consacré à XUL en est le meilleur exemple. Ainsi le succès de Firefox repose déjà naturellement plus sur les innovations offertes à l'utilisateur, que sur son aptitude à faire aussi bien que ses concurrents. Mais que dire alors du framework Mozilla dont Firefox et XUL ne sont que des parties émergées ?

Ici il n'est donc plus question de suiveurs : le framework Mozilla livre dès aujourd'hui une technologie que Microsoft ne proposera qu'avec son prochain système, au mieux dans un an. En outre, ce framework est, lui, d'ores et déjà multi-plates-formes !

En vous accompagnant dans la création d'une application type, Jonathan vous propose de découvrir les facettes parfois complexes d'un environnement novateur qui a fait le choix des standards et d'une réutilisation maximale des langages existants.

Car c'est une autre qualité du framework Mozilla que de tirer profit des standards ouverts, sur lesquels nous avons tous, « techniciens » et entreprises, investi ces dernières années. XML y joue naturellement un rôle clé, mais l'environnement reprend aussi largement CSS, JavaScript et l'incontournable DOM, désormais bien entré dans les mœurs et qui ouvre la voie à la mutualisation des efforts...

Aussi, que vous ayez déjà capitalisé une expérience en PHP 5 (et son API DOM désormais conforme) ou en JavaScript côté client (avec Firefox, voire Microsoft Internet Explorer), vous ne serez pas dépaycé à la lecture de cet ouvrage. De même, si vous avez créé des applications AJAX, vous pourrez réutiliser votre savoir-faire pour dynamiser votre application XUL.

Mozilla réussit ainsi le tour de force de créer un environnement novateur tout en capitalisant sur l'existant, qualité déterminante face à ses concurrents. Il est temps pour vous de l'expérimenter ! :-)

Stéphane MARIEL

Table des matières

AVANT-PROPOS	XI
1. INTRODUCTION À XUL	1
Mozilla et XUL • 2	
Un point de départ : le code source de Netscape • 2	
Des problèmes d'évolution • 3	
Un nouveau moteur et une nouvelle architecture • 3	
Un projet ambitieux qui a réussi • 4	
La place de XUL au sein du XPFE • 6	
XUL : un langage qui ne s'utilise pas seul • 7	
Les technologies connexes à XUL, CSS et JavaScript • 8	
Organisation générale du XPFE • 9	
Pourquoi choisir XUL ? • 11	
Un langage facile à apprendre • 11	
Portable • 12	
Sans limite ! • 12	
En résumé... • 13	
2. L'ÉTUDE DE CAS	15
Introduction • 16	
Fonctionnement • 17	
Principe de fonctionnement d'un forum • 17	
Le fonctionnement de XUL Forum • 18	
Les différents modules • 18	
L'inscription • 18	
L'identification • 18	
L'écran principal • 19	
La fenêtre d'ajout/modification/lecture d'un sujet • 20	
Les différentes parties du XPFE mises en jeu • 21	
Les technologies « statiques » • 21	
Les technologies « dynamiques » • 22	
Quels seront les avantages de XUL dans ce cas ? • 22	
Des modules réutilisables • 23	
Une approche croissante des difficultés • 23	
Une large utilisation des possibilités du XPFE • 23	
Un environnement professionnel • 23	
Points noirs à l'utilisation de XUL • 24	
En résumé... • 25	
3. PREMIERS PAS EN XUL	27
Un premier fichier XUL • 28	
À la découverte des premiers éléments... • 31	
Première version de l'écran d'authentification • 32	
Corriger les premières erreurs : utilisation de boîtes • 32	
Un élément plus adapté : un tableau • 34	
La touche finale : spacers • 35	
En résumé... • 37	
4. UNE VÉRITABLE EXTENSION MOZILLA.....	39
La séparation générale des fichiers • 40	
Contents.rdf et dossier content • 41	
Modification du fichier chrome.rdf • 43	
Intégration d'une DTD et début de l'internationalisation • 45	
Dossier locale • 47	
Modification du fichier XUL et ajout d'une DTD • 48	
Chrome.rdf • 50	
En résumé... • 51	
5. XUL AVANCÉ : LA FENÊTRE PRINCIPALE.....	53
La structure globale • 54	
Découpage avec les principales boîtes • 54	
Séparation en overlays • 56	
Dans le document principal • 56	
Dans les fichiers overlay • 58	
Le fichier index-barres-overlay.xul : barres d'outils, de menu et de statut • 61	
Les menus • 62	
Des menus plus évolués • 64	
Éléments communs à toutes les pages • 66	
La barre d'outils • 69	
La barre de statut • 70	
Résumé : les barres • 71	
Les overlays : arbre et onglets • 72	
L'arbre • 73	
La liste des membres • 75	
En résumé... • 77	

6. PERFECTIONNEMENT DU RENDU AVEC CSS..... 79

- Présentation de CSS ; utilisation dans Mozilla • 80
- Débuts avec CSS : effets sur du texte • 82
 - Mise en place de CSS • 82
 - Premiers effets sur du texte • 84
- Retoucher le positionnement avec CSS • 90
- CSS spécifique à Mozilla • 91
 - La barre d'outils • 91
 - Autres propriétés CSS • 93
 - Les onglets du panneau de gauche • 93
 - Couleur des lignes de l'arbre • 94
 - Images dans les menus • 95
 - Utilisation d'une propriété propre à Mozilla • 96
- En résumé... • 96

7. PREMIÈRE ANIMATION DE L'INTERFACE AVEC JAVASCRIPT 99

- Concepts de base du langage • 100
 - Syntaxe de base • 101
 - Variables, utilisation de fonctions • 101
 - Commentaires • 101
 - Chaînes • 102
 - Déclarations de fonctions • 102
 - Méthodes d'objets • 103
 - Tableaux • 103
 - Objets : instanciation • 103
 - Exceptions • 104
 - Plus... • 104
- Intégration à XUL • 105
- Application directe à XUL Forum • 106
 - Une première routine pour l'affichage d'erreurs • 106
 - Multi-langue avec l'objet stringbundle • 109
 - Plus de manipulation DOM : options avancées à la connexion • 110
 - Le code servant à montrer les options avancées • 112
- Communication avec l'extérieur : récupération d'un fichier de configuration • 114
 - L'objet XMLHttpRequest • 114
 - L'analyse avec DOM • 118
- Approche des composants XPCOM : fonction include() • 120
 - Les composants XPCOM • 120
 - Notre inclusion • 121
- En résumé... • 123

8. AUTOMATISATION AVEC RDF..... 125

- Le format RDF : explications détaillées • 126
 - Les nœuds et les arcs • 126
 - Nœuds et URI • 127
- Sérialisation de RDF avec RDF/XML • 128

Listes • 131

Génération de RDF avec PHP • 132

- Objectifs • 132
- Le fichier PHP • 133

Retour à XUL • 136

- Un premier modèle simple • 136
- Un modèle plus complexe • 138
 - Modification côté PHP • 138
 - Exploitation côté XUL • 139

Amélioration de RDF avec JavaScript • 142

- Version synchrone • 143
- Version asynchrone • 146

En résumé... • 148**9. UNE INTÉGRATION AU CŒUR DE MOZILLA 151**

- Extension de l'interface du navigateur avec de nouveaux overlays • 152
 - Modification du fichier contents.rdf • 152
 - Où trouver les fichiers à modifier ? • 153
 - Gestion multiple : Firefox, Thunderbird, Mozilla • 155
 - La suite Mozilla • 156
 - Le navigateur Firefox • 157
 - Le client mail Thunderbird • 159
- Utilisation des préférences • 160
 - Présentation • 160
 - Les fonctions XPCOM essentielles • 161
 - Le code de XUL Forum • 161
 - Application à l'identification • 163
- Autres techniques utiles • 165
 - Raccourcis clavier • 165
- En résumé... • 167

10. JAVASCRIPT VERSION « PRO » : LDAP..... 169

- Recherches LDAP avec JavaScript et nos propres composants XPCOM • 170
 - La structure LDAP de XUL Forum : le DIT • 172
 - La succession des différentes fonctions • 173
 - L'initialisation • 175
 - Nos propres composants XPCOM en JavaScript : listeners • 176
 - Les composants • 176
 - Création d'une file d'attente • 179
 - Obtention d'une opération • 180
- Identification avec un simple bind • 180
- Obtenir la liste des membres • 182
 - Analyse du côté LDAP • 182
 - Traitement XUL • 184
- Les informations d'un connecté • 185
- En résumé... • 187

11. DO-IT-YOURSELF WIDGETS : XBL..... 189**Fonctionnement d'un widget XBL • 190**

Le widget fenetreMsg • 190

Notre implémentation : le binding fenetreMsg • 192

Le contenu du widget : <content> • 193

Le widget vu de l'extérieur : un bloc div et une classe CSS • 193

Le contenu intérieur : mélange HTML et XUL • 194

La mise en forme CSS • 196

Un widget qui s'anime : <implementation> • 198

Les propriétés et les champs • 198

Les méthodes • 199

Plier • 200

Déplier • 201

Cacher • 202

Un widget réactif : le <handler> • 202

Le double-clic • 203

Le bouton de la souris est baissé • 204

La souris est déplacée • 206

La souris est relâchée • 206

Mise en relation avec la page principale de XUL Forum • 206

Modifications dans le fichier XUL • 207

Modifications dans le JavaScript • 207

En résumé... • 210

**12. LES SERVICES WEB, POUR UNE COMMUNICATION
HARMONIEUSE ENTRE CLIENT ET SERVEUR..... 213****SOAP en détail : application à XUL Forum • 215****Le serveur en pratique avec PHP5 • 217**

Le serveur, vue globale • 217

Permettre l'authentification de l'utilisateur • 219

Lire un message • 220

Enregistrer un message • 221

Le client JavaScript • 223

L'initialisation • 223

Est-on en phase avec le serveur ?

Vérification de la session • 224

Pas de duplication de code : une routine pour tous nos appels SOAP • 225

Le moment crucial : l'identification • 227

Indispensable : lire un message • 227

Vital : poster nos propres messages • 228

Les modifications de l'interface avec XUL • 229

Changements dans le fichier XBL • 229

Changements dans les fichiers JavaScript • 233

En résumé... • 235

13. DISTRIBUTION DE XUL FORUM AVEC XPINSTALL 241**Le fichier xulforum.xpi • 242**

Création du fichier xulforum.jar • 242

install.rdf : comment installer XUL Forum ? • 243

Compatibilité avec Mozilla 1.x : install.js • 247

Signaler des mises à jour futures • 249

En résumé... • 251

LE MOT DE LA FIN 253**A. LE FUTUR : VERS FIREFOX 1.5 ET AU-DELÀ 255**

Il n'y a plus de contents.rdf dans Firefox 1.5 et

Thunderbird 1.5 • 256

Un petit nouveau... XUL Runner ! • 258

Une valeur sûre : SeaMonkey • 261

De nouvelles voies : SVG et <canvas> • 262

Des modifications mineures • 263

B. LISTE DES COMPOSANTS XPCOM UTILISÉS 265**C. LA LISTE DES FICHIERS DE L'APPLICATION 269****D. CSS : SYNTAXE, SÉLECTEURS, PROPRIÉTÉS 273**

Syntaxe de base de CSS • 274

Les sélecteurs • 276

Les propriétés utiles en CSS 2.1 • 277

Les extensions Mozilla à CSS • 281

E. RÉFÉRENCE DES ÉLÉMENTS XUL GRAPHIQUES 285**INDEX 301**

Avant-propos

Un renouveau nécessaire dans le monde des clients riches

Le grand réseau Internet arrive à un tournant majeur de son existence : derrière lui, une balkanisation effroyable des sites, des normes et des navigateurs ; devant lui, une perspective d'évolutions prometteuses, de nouvelles bases de développement et de nouveaux standards. On entend désormais parler de nouveaux formats, comme XML, CSS, XHTML, ECMAScript et de nouvelles pratiques de développement plus ordonnées, posant des fondations solides pour l'avenir et étendant le champ fonctionnel des applications fondées sur les technologies de l'Internet.

Dans le même temps, on assiste à un renouveau des clients riches, applications pour les utilisateurs, souvent programmées dans des langages lourds associés à des bibliothèques graphiques. Java et Swing, C et GTK ou QT animent ainsi un grand nombre de postes de travail. En dépit de leur succès, ces langages réclament un long apprentissage et obligent parfois à ce que les programmes qui en résultent soient adaptés pour Linux, Windows ou Mac OS, engendrant des coûts de développement élevés et des évolutions ralenties.

Que faire alors ? N'existe-t-il pas une perspective visant à associer vivacité des applications web et richesse des applications *desktop* ?

Une réponse innovante est née au sein du projet Mozilla : elle s'appelle XPFE, Cross Platform Front-End et est constituée d'un ensemble de technologies, qui reposent toutes sur les standards du Web et sont toutes aussi puissantes qu'un langage traditionnel pour applications riches. XUL, objet de cet ouvrage, représente la partie visible de ce grand ensemble, puisque, associé à des technologies complémentaires comme JavaScript ou CSS, pour ne citer que les plus connues, il se charge de décrire les interfaces graphiques.

Quel est l'objectif de cet ouvrage ?

L'objet qui sous-tend tout l'ouvrage est de démontrer par l'exemple que le XPFE n'a rien à envier à un environnement de développement traditionnel et que, de surcroît, il est capable de dépasser les plus grands, tant par son caractère évolutif et la facilité avec laquelle on le maîtrise, que par sa simplicité et la rapidité de développement qu'il entraîne. L'exemple d'application qui illustre ce livre démontrera l'immense étendue des possibilités des technologies conçues au sein du projet Mozilla : cette application s'appelle XUL Forum et se compose, comme le nom le laisse présager, d'un système de forum écrit en XUL !

Les solutions techniques choisies pour ce développement ne seront pas nécessairement les plus simples : nous avons privilégié celles qui pourront vous servir plus tard pour sortir de problèmes bien plus complexes dans vos propres applications ! Parfois, nous utiliserons des solutions relativement peu élégantes, mais pratiques et fonctionnelles. S'il n'est pas toujours possible de détailler la solution idéale, des apartés seront là pour replacer la technologie dans son contexte. Notes et annexes illustreront la diversité des technologies XML, CSS ou JavaScript sans, espérons-le, alourdir le propos principal.

Cet ouvrage est donc à la fois efficace et précis. L'objectif final est d'arriver au résultat qui sera décrit dans le deuxième chapitre, une fois qu'aura été parcourue la vaste étendue des technologies Mozilla utiles à notre projet.

À qui s'adresse cet ouvrage ?

On pourrait croire à tort que l'ouvrage est réservé aux programmeurs confirmés : il n'en est rien ! Le développeur web ayant déjà essayé JavaScript et HTML, le codeur confirmé familier du C et du XML, le *designer* rodé à CSS, tous pourront s'instruire à travers l'application pro-

posée en exemple. Chacun y trouvera son intérêt et pourra s'initier à des techniques qui, même s'il ne les utilise pas dans le cadre de Mozilla, lui resserviront plus tard.

Après avoir vu JavaScript au travers de cinq chapitres, puis maîtrisé XML ou compris les astuces de CSS, vous serez donc capable de reproduire nos modèles et d'utiliser les connaissances acquises, par exemple, pour votre site personnel ou celui de votre entreprise.

L'ouvrage est également accessible aux débutants : l'approche croissante des difficultés et les nombreux apartés leur permettront de ne pas se perdre.

Structure de l'ouvrage

Les Cahiers du programmeur suivent une structure très pragmatique : les technologies seront abordées au fur et à mesure du développement d'un cas concret, lorsque cela sera opportun. Nous commencerons donc par la technologie XUL, pour finir par XPInstall, permettant de distribuer l'application une fois finie.

Le **chapitre 1** présentera en détail les motivations à l'origine de la naissance de XUL, la structure du XPFE, sa place au sein du projet Mozilla et les différentes briques qui le composent.

Le **chapitre 2**, écho concret du premier, décrira l'étude de cas et le rôle de chacune des technologies mentionnées précédemment dans XUL Forum.

Dans le **chapitre 3**, nous entrerons dans le vif du sujet avec la création d'un premier fichier .xul !

Pour être plus à notre aise, nous nous installerons directement dans l'environnement de Mozilla : ce sera le **chapitre 4**, celui où nous créerons concrètement une extension.

Mais il ne faut pas négliger XUL pour autant : nous nous attaquerons aux parties les plus ardues de l'interface dans ce **chapitre 5**.

Une application professionnelle l'est souvent grâce aux petits détails : c'est le rôle de CSS, qui permet de peaufiner l'interface, qui sera abordé dans le **chapitre 6**.

Mais tout ceci est bien trop statique ! Pour animer les différents composants, JavaScript nous viendra en aide dans le **chapitre 7**.

Avec le **chapitre 8**, c'est du côté du serveur que nous irons, pour voir comment engendrer du contenu directement exploitable par XUL, grâce au modèle RDF.

Le **chapitre 9** sera l'occasion d'une pause après deux chapitres intenses : nous verrons comment tirer parti des avantages offerts aux extensions par Mozilla.

Le **chapitre 10** sera un peu particulier : il présentera l'interaction entre Mozilla et les services d'annuaire LDAP. Il vous sera possible de ne pas le lire si vous êtes trop usé par les précédents et si tout cela ne vous tente guère : dans la version finale et testable sur le site du projet, ce mécanisme ne sera pas présent. Si, par contre, vous êtes en entreprise, si vous avez l'habitude de cette technologie, ou si, tout simplement, vous en êtes curieux, alors n'hésitez pas, dévorez-le !

Le **chapitre 11** sera en quelque sorte un condensé de tout ce qui a été vu précédemment : XBL, ou comment créer vos propres éléments graphiques (*widgets*).

Le **chapitre 12** apportera plus de communication entre applications par le biais des services web, dialogue réciproque entre client et serveur.

Enfin, puisque tout vise à être fonctionnel, nous pourrions distribuer notre application grâce au **chapitre 13**, en l'empaquetant et en rédigeant les fichiers décrivant la procédure d'installation. Ce sera alors une « vraie » extension .xpi !

Les différentes annexes permettront d'approfondir plus encore votre expérience XUL :

- Les modifications attendues de la prochaine version 1.5 de Firefox et XULRunner.
- Les composants XPCOM utilisés, ainsi que ceux qui pourront servir un jour.
- La liste complète des fichiers de l'application, pour ne pas s'y perdre !
- Les références XUL et CSS, pratiques en cas d'oubli.

POUR ALLER PLUS LOIN

Les différentes versions de XUL Forum

Vous retrouverez souvent au fil du texte des remarques intitulées « Pour aller plus loin ». Elles ouvrent généralement la voie à des améliorations possibles du code, mais non décrites dans le livre. Beaucoup de ces améliorations seront disponibles sur xulforum.org : il y aura en effet trois versions du code source. La première sera le code tel que le lecteur pourra le trouver à la fin du chapitre 13. La seconde sera le code amélioré, avec la plupart des possibilités « Pour aller plus loin » intégrées. Ces deux versions intégreront le support LDAP. Une dernière version supprimera ce support, transformant ainsi XUL Forum en application fonctionnelle et installable en tant que xpi.

Pour aller plus loin

Une fois que vous aurez lu ce Cahier, vous pourrez tester directement XUL Forum à l'adresse www.xulforum.org, récupérer des extensions adaptées à votre version du navigateur et même les codes source décrits dans le livre ! Le forum sera d'ailleurs une très bonne occasion d'en parler...

À l'issue de cette lecture, vous pourrez manier sans problème les principaux langages qui formeront le Web de demain et développer de superbes clients riches en les couplant à Mozilla !

Jonathan PROTZENKO

jonathan@xulforum.org

chapitre

1



Introduction à XUL

Les applications web sont, depuis le développement de l'Internet, une solution de choix dans la conception de nouvelles applications, au même titre que les clients lourds écrits en C ou en C++.

Comment XUL permet-il de développer une application dans un contexte web tout en combinant des avantages inaccessibles à du simple HTML ?

SOMMAIRE

- ▶ Mozilla et XUL : une histoire intimement liée
- ▶ La place de XUL dans le *framework* Mozilla
- ▶ Les avantages de XUL

MOTS-CLÉS

- ▶ XPFE
- ▶ XPCOM
- ▶ Portabilité

Depuis le développement spectaculaire qu'a connu l'Internet dans la dernière décennie, deux grands types de solutions s'offrent au développeur pour la conception de son application graphique. Une application web, écrite en HTML, combinant des CSS et du JavaScript, permet d'achever le projet rapidement sans exiger du développeur des compétences longues et difficiles à maîtriser. Cependant, il faut faire appel à un client lourd (écrit en Java, en C, etc.) si l'on veut que l'utilisateur puisse effectuer des actions complexes, comme interagir avec le système de fichiers, disposer des fonctions de glisser/déposer, ou même utiliser des éléments graphiques plus riches. L'emploi de telles techniques est bien sûr plus agréable pour l'utilisateur, mais plus difficile à mettre en œuvre pour le développeur.

XUL apporte une réponse à ce dilemme : fondé sur les nouvelles technologies du Web, il permet d'aller beaucoup plus loin que les simples sites en HTML, sans pour autant nécessiter un apprentissage difficile ou une mise en œuvre complexe. Nous allons essayer de comprendre comment ce langage a été créé et pourquoi il peut apporter une solution pour le développement de nouvelles applications riches.

Mozilla et XUL

Par ellipse, on parle communément de « XUL », en oubliant que ce langage est utilisé avec d'autres technologies et, surtout, qu'il fait partie intégrante de ce que l'on peut appeler le *framework* Mozilla. Il ne peut en être détaché, c'est pourquoi un petit historique s'avère nécessaire. Comment et pour quelles raisons XUL a-t-il fait son apparition dans ce qui ne devait être au départ qu'un navigateur web ?

HISTORIQUE Mozilla, pourquoi ce nom ?

Mozilla était le nom interne utilisé par Netscape pour désigner son navigateur. Mozilla est la contraction de « Mosaïc Killer », Mosaïc étant le premier navigateur Internet grand public. L'objectif de Netscape était justement de le détrôner.

Un point de départ : le code source de Netscape

L'histoire du projet Mozilla, débute en mars 1998, lorsque Netscape décide d'ouvrir une grande part du code de la suite *Netscape Communicator* à la communauté Open Source : le communiqué de presse annonce une disponibilité du code sous une licence libre, pour profiter de « la puissance créatrice de nombreux développeurs [...] qui apporteront leurs améliorations ». Un petit groupe de passionnés se constitue alors autour de ce code de base et se charge de coordonner les efforts venant à la fois de Netscape et de la communauté du Logiciel Libre.

Des problèmes d'évolution

Cependant, le code de Netscape se révèle vite difficile à maintenir. Par exemple, la prise en charge des éléments `<frameset>` ou `<optgroup>` de HTML 4.0 requiert un travail parallèle pour chaque environnement supporté : Microsoft Windows, Linux ou Mac OS. Tous ces inconvénients ont décidé les développeurs Mozilla à abandonner le moteur de rendu original au profit du nouveau moteur développé par les équipes de Netscape appelé *new generation layout* ou NG Layout.

Même si le nom du projet est demeuré « NG Layout », le moteur est ensuite devenu Gecko. Ses développeurs se sont donné pour but de le rendre plus puissant, plus léger et plus rapide que son prédécesseur. La première version officielle a été publiée par Netscape en 1998 et pouvait tenir sur une disquette ! Gecko symbolise l'influence de la communauté Open Source : il a été dès le départ pensé pour être compatible avec les standards du Web et même si Netscape ne l'a pas adopté tout de suite, il reflète une volonté de la part du monde du Logiciel Libre d'avoir enfin un navigateur respectueux des recommandations du W3C.

HISTORIQUE Archives de Mozilla.org

Cette page est consacrée à l'ancien moteur de rendu. Si vous lisez la page relative au travail à effectuer pour supporter HTML 4.0, vous comprendrez pourquoi les développeurs ont voulu changer la structure interne de Mozilla ! Implémenter l'élément `<iframe>` était estimé à trois semaines de travail...

► <http://www.mozilla.org/classic/layout-classic/>

B.A.-BA Moteur d'un navigateur

Le moteur de rendu est la partie du navigateur qui se charge de « dessiner » la page web, c'est-à-dire de tracer les éléments de formulaire ou d'afficher les images au bon endroit. Plus généralement, le moteur de Mozilla combine à cette fonction de rendu l'analyse du document (appelée le *par-sing*), la gestion des styles, ou l'accès à la structure du document via le DOM (Document Object Model). Il a de plus été conçu pour être embarqué dans des applications tierces.

Un nouveau moteur et une nouvelle architecture

Toujours dirigé par Netscape, le projet Mozilla s'est également trouvé confronté à un autre problème : une suite constituée d'un navigateur, d'un client de courrier électronique et d'un éditeur HTML ne pouvait reposer sur le seul moteur de rendu. Autrement dit, même si Gecko était un projet très prometteur et sur lequel Mozilla fondait ses espoirs, la modularité et l'évolutivité de la suite n'était pas au rendez-vous. Là encore, l'organisation du code de Netscape et la difficulté de le faire évoluer, même avec le NG Layout, ont conduit l'équipe de développement à repenser totalement Mozilla. Ils ont conçu un projet plus ambitieux encore : réaliser une plate-forme générique de développement d'applications capable d'accueillir un client de courrier, un éditeur HTML et, naturellement, un navigateur. Cette réorganisation du code est devenue le XPFE : Cross Platform Front-End.

CULTURE Citation

La citation toujours mentionnée dans un livre sur XUL se trouve à cette adresse. Elle explique pourquoi les développeurs ont décidé d'abandonner l'ancien système de rendu : « In the beginning, there were three frontends (...). Each took a suite of developers to maintain... ».

► <http://www.mozilla.org/xpfe/ElevatorTouchyFeely.html>

HISTORIQUE Roadmap fin 1998

Voici la *roadmap*, autrement dit les orientations définies pour le projet Mozilla en octobre 1998. Ce document marque un tournant radical dans la politique de développement : c'est en effet à cette date qu'a été prise la décision d'abandonner l'ancien code de Netscape et de se reposer désormais sur XPFE et Gecko.

► <http://www.mozilla.org/roadmap/roadmap-26-Oct-1998.html>

XPFE, couplé au moteur Gecko, représente donc la réponse des développeurs Mozilla à tous les problèmes rencontrés auparavant. Conçu dès le départ pour être multi-plates-formes, XPFE, une fois achevé, deviendrait la solution rêvée des développeurs !

Un projet ambitieux qui a réussi

Ce projet, accueilli parfois avec scepticisme, a tenu ses objectifs : Mozilla s'est détaché de Netscape suite au rachat de ce dernier par AOL. C'est maintenant la Mozilla Foundation, organisation à but non lucratif, qui gère le projet, en apportant un support financier et un cadre légal pour faire vivre le lézard rouge, emblème de Mozilla.

Entre temps, des *milestones* (sorties du logiciel précédant une version importante, destinées aux dernières phases de test) se sont succédées avant de voir enfin apparaître en décembre 2000 la première « vraie » version de Mozilla, la version 0.6, sous le nom de code *SeaMonkey*. Mozilla en est maintenant à sa version 1.8 et la fondation accueille d'autres projets parallèles : des navigateurs alternatifs, un projet d'agenda, un client de courrier électronique indépendant... et le site mozilla.org est toujours aussi actif !

ATTENTION Mozilla et SeaMonkey

Dernièrement, la fondation Mozilla a décidé d'abandonner le support « officiel » du produit « Mozilla Suite ». La suite Mozilla est donc devenue *SeaMonkey* (repreant le nom de code original), un projet géré par la communauté et profitant du soutien logistique de la Fondation Mozilla (machines de test, serveur cvs...). La Fondation Mozilla n'assure plus que les mises à jour de sécurité de la branche 1.7 et ce qui devait être la branche 1.8 sera SeaMonkey 1.0, le projet communautaire.

Dans cet ouvrage, commencé alors que SeaMonkey n'avait pas encore vu le jour, c'est surtout le terme de « Suite Mozilla » qui sera employé. En annexe A, vous retrouverez une copie d'écran de SeaMonkey.

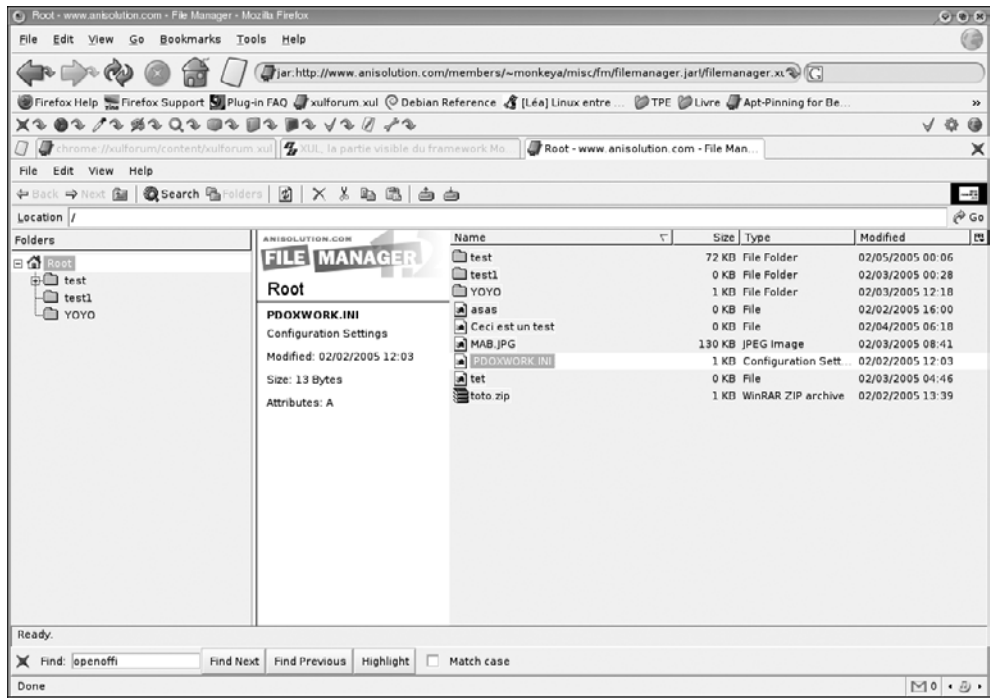


Figure 1–1
<http://filemanager.mozdev.org/>
vous propose une interface de gestion de fichiers similaire à l'explorateur de Windows

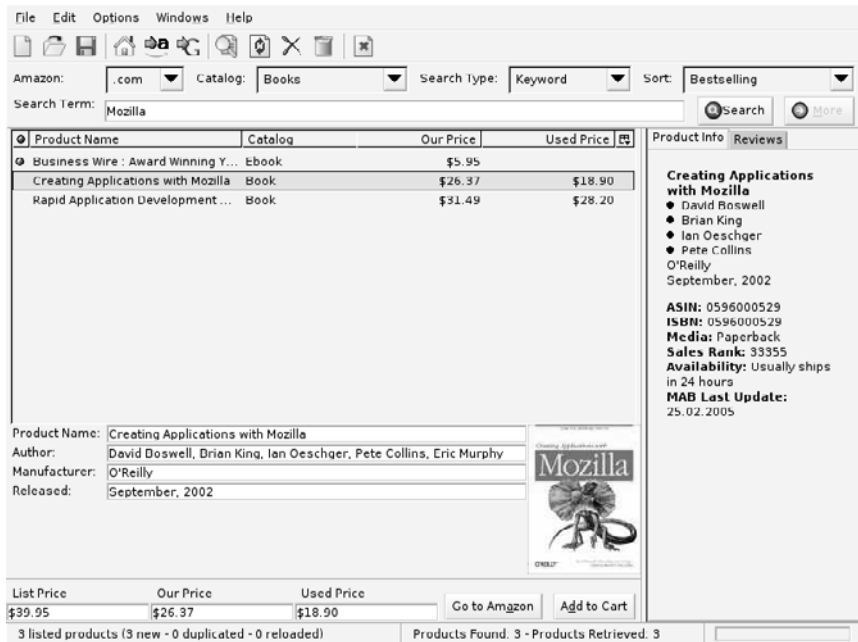


Figure 1–2
Mozilla Amazon Browser, une autre application souvent citée en exemple, permet de consulter le site de vente en ligne Amazon via une interface XUL

Maintenant, Gecko est intégré dans de nombreuses applications : Firefox, Sunbird (projet de calendrier autonome), Thunderbird, mais aussi un éditeur XHTML comme NVU ou encore Galeon, un navigateur intégré à l'environnement de bureau GNOME. L'architecture XPFE est stable et sert de plate-forme de développement pour tous ceux qui veulent créer des clients riches en utilisant les standards du Web et la puissance d'un logiciel Open Source : on pourrait citer en exemple le *File Manager* de la société Anisolution qui est le « sommet de l'art en XUL » : la copie d'écran parle d'elle-même.

CULTURE **Gecko là où on ne l'attend pas**

Deux exemples d'utilisation exceptionnelle de Gecko, preuve qu'il ne faut pas réduire le célèbre moteur de rendu à un contexte web :

- MozillaZine (le magazine sur l'actualité de Mozilla) annonce une version serveur (non graphique) pour Gecko : à partir de documents XHTML, le moteur rend un fichier PostScript (.ps), en utilisant le même mécanisme que pour une impression classique. Ce fichier PS peut ensuite être converti en PDF, JPEG...
 ▶ <http://www.mozillazine.org/talkback.html?article=6927>
- SymphonyOS, une distribution Linux fondée sur la distribution Knoppix (elle-même fondée sur Debian), a décidé d'axer son développement sur la meilleure expérience utilisateur possible. Elle a pour ce faire développé son propre « *desktop environment* » et son propre environnement applicatif... conçu sur Mozilla ! On retrouve bien sûr XUL, CSS, JavaScript... qui servent à créer un bureau de conception très originale et permettent le développement d'applications tierces comme un panneau de contrôle.
 ▶ <http://symphonyos.com/mezzo.html>
 ▶ <http://symphonyos.com/orchestra.html>

La place de XUL au sein du XPFE

Mozilla s'est donc progressivement transformé : d'une simple réécriture de *Netscape Communicator*, le projet est devenu un *framework* applicatif, un environnement pour accueillir des applications. Si certaines de ces applications sont bien connues, tels le client mail, le module de *chat* IRC intégré appelé ChatZilla ou l'éditeur HTML, sans oublier le navigateur lui-même, le *framework* XPFE vous permet de créer vos propres applications destinées à être utilisées avec Mozilla. C'est ici que XUL intervient puisqu'il est l'un des différents langages utilisés au sein du XPFE.

XUL : un langage qui ne s'utilise pas seul

XML est utilisé pour décrire la structure d'un document. XUL, pour *XML User interface Language*, sera utilisé pour décrire la structure de l'application. Le parallèle entre les deux est très proche : on pourrait dire que XUL est une application du XML aux interfaces graphiques.

Il faut bien comprendre le rôle de XUL : c'est un langage qui se charge de la construction de l'interface graphique. XUL seul ne permet de réaliser que la structure de l'application : le placement des différents éléments, leurs attributs et leur organisation spatiale permettent de créer une application statique. Pour rendre l'application vivante et permettre à l'utilisateur d'interagir avec elle, il faut décrire les actions et les comportements. Ces aspects ne sont pas pris en compte par XUL, mais reposent sur un langage de *script*.

Exemple francisé montrant la logique XUL

```
<fenêtre>
  <boîte>
    <zonedetexte nom="maZoneDeTexte" />
    <bouton titre="Voici mon bouton !" />
  </boîte>
</fenêtre>
```

B.A.-BA XML

XML signifie eXtensible Markup Language, autrement dit, un langage à balises. XML n'est pas en lui-même un langage, c'est plutôt un format d'échange de données. On parle alors de méta-langage : un ensemble de règles qui permettent de définir un langage. XHTML, reformulation de HTML selon XML, est un exemple de langage défini à partir des règles de XML.

```
<?xml version="1.0" ?>
<cahiersduprogrammeur>
  <livre sujet="XUL" auteur="Jonathan" />
  <livre sujet="PHP5" auteur="Stéphane" />
</cahiersduprogrammeur>
```

XML est par définition lisible par un humain : les balises (ou *tags* en anglais) tels que `<livre />` parlent d'elles-mêmes. Il est bien sûr possible de produire du XML incompréhensible en compliquant sa syntaxe, mais alors pourquoi ne pas utiliser tout simplement un format propriétaire ?

XML est une recommandation du W3C. Il est facilement utilisable par les développeurs car il existe des fonctions de manipulation XML pour tous les langages : par exemple, la bibliothèque libxml2 sert au C, mais on en trouve aussi pour PHP, Java, JavaScript ou Perl.

On parlera d'élément pour désigner une balise et son contenu (exemple : `<livre> La technologie XUL... </livre>`). Le nom des balises est entièrement libre et chacun peut choisir de définir son fichier XML comme bon lui semble. Cependant, XML obéit à quelques règles strictes :

- une balise ouverte `<livre>` doit toujours être fermée, soit par `</livre>`, soit directement en s'écrivant `<livre />` ;
- les attributs sont en minuscule : `<livre auteur="" />` et leur valeur est toujours placée entre guillemets doubles ;
- un encodage doit être spécifié, ou, par défaut, le fichier doit être encodé en Unicode.

Ce langage de balises (et non pas langage de programmation !) sera utilisé durant tout le long du livre. Nous verrons plus en avant qu'il existe des moyens de définir la structure d'un fichier XML (les balises que l'on doit y retrouver, leur ordre, les attributs qu'elles proposent), comme une DTD ou un Schéma XML.

Pour plus de détails, voir :

 *XML, la référence*, aux éditions O'Reilly.

Le choix des développeurs s'est porté sur un autre grand standard du W3C : ECMAScript, norme dont l'implémentation la plus connue est le JavaScript. Ainsi on pourra positionner une zone de texte avec XUL et valider son contenu avec JavaScript.

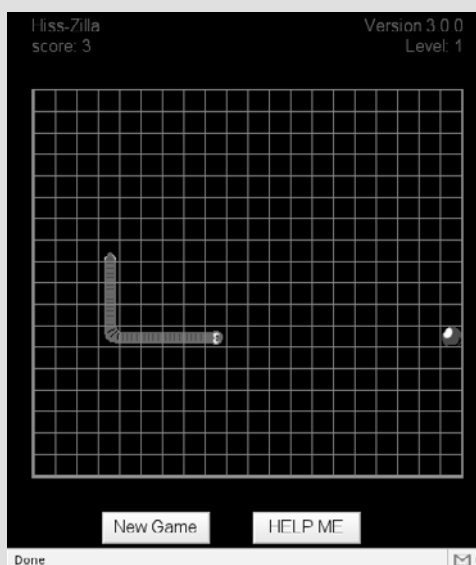
Il est aussi intéressant de pouvoir agir sur l'apparence de l'application : taille ou format du texte sont, par exemple, des attributs que l'on ne doit pas définir avec XUL, qui ne doit s'occuper que de la structure du document. Ainsi, un autre standard quasi indispensable dans une application XUL est CSS (Cascading Style Sheets). CSS vous permettra de contrôler l'aspect de votre application, les couleurs, la taille des éléments, les images notamment, mais aussi d'autres composants plus complexes que vous découvrirez tout au long de ce livre.

CULTURE Utilisations de XUL

XUL est donc au cœur d'une plate-forme pour le lancement d'applications. Dans les dernières versions *milestone* de Mozilla, une application « console » était livrée avec le navigateur. Codée en XUL, elle implémentait quelques fonctions de base comme la commande `ls` du *shell*. Ceci visait à montrer que Mozilla n'était pas limité au contexte web. De même, vous pouvez jouer en XUL si vous le désirez avec les jeux de :

► <http://games.mozdev.org>

Figure 1-3 Un serpent en XUL !



B.A.-BA RDF

RDF signifie Resource Description Framework. C'est un standard qui permet de produire des documents XML contenant du sens : là où une page HTML ne contient que du texte sans organisation, un document RDF a un contenu structuré, qui permet à une machine d'analyser le document et de lui associer une structure logique. Ceci n'est qu'un bref aperçu de RDF, nous le détaillerons au chapitre qui y est consacré.

Les technologies connexes à XUL, CSS et JavaScript

Mozilla est certainement sur le marché un navigateur parmi les plus respectueux des standards, c'est pourquoi il est possible et même recommandé d'en tirer pleinement parti. Comme les fichiers XUL sont au départ des fichiers XML, vous pourrez utiliser diverses DTD pour définir des entités qui serviront à l'internationalisation, ou bien encore RDF. RDF (voir ci-contre) peut être utilisé pour créer automatiquement du XUL, accéder à des données internes comme les favoris internet, l'historique des sites consultés, etc. Nous en verrons plus au chapitre consacré à ce sujet.

D'autres langages fondés sur XML font également partie du XPFE : nous en découvrirons un appelé XBL plus loin dans ce livre.

B.A.-BA DTD

DTD ou Document Type Definition est un standard qui permet de décrire la structure d'un document XML. La validation d'un document XML permet de voir s'il est conforme à sa DTD. Il existe, par exemple, des DTD pour les différentes versions du XHTML. Pour XUL, une DTD servira surtout à définir des entités. En XHTML, une entité est par exemple ´ . Avec XUL, une entité sera &titreDeMaFenetre; et sera « *My Window* » ou « *Ma Fenêtre* », selon la langue : nous approfondirons ce concept plus loin.

Organisation générale du XPFE

Après avoir vu les technologies liées au XPFE, nous allons voir la structure interne du XPFE, autrement dit comment s'organisent les différentes parties de ce grand ensemble (voir figure 1–4, page suivante).

Au centre du XPFE se trouve XUL. Il est intimement lié à JavaScript : les événements créés par XUL appellent du JavaScript et vice versa, une fonction JavaScript peut modifier la structure du document XUL. Les interactions JavaScript/XUL se font par l'intermédiaire du DOM (Document Object Model). Le rendu d'un document XUL est quant à lui à la charge de Gecko, le moteur de Mozilla : ce dernier analyse le document XUL, le dessine sur le système d'affichage du client et prend également en charge les modifications apportées au document XUL via DOM.

Ceci constitue en quelque sorte la partie visible de l'iceberg qu'est Mozilla : ce sont les éléments que manipule directement le développeur lorsqu'il utilise XUL. Cependant, Mozilla n'est pas programmé entièrement en JavaScript. Enregistrer les préférences de l'utilisateur sur le disque n'est pas une tâche directement réalisable en JavaScript. C'est pourquoi Mozilla dispose de parties en code natif, utiles pour se connecter à un serveur distant par exemple. Ainsi, si vous essayez d'obtenir des informations sur une méthode pour appeler une fonction SOAP, vous obtiendrez une fonction de type *native code*. Ce sont des *composants XPCOM*. Un composant XPCOM possède une interface fixe, qui sert de point de contact pour les autres langages et son implémentation peut varier en fonction de la plate-forme. Pour se connecter à un serveur via les *sockets*, il existe des interfaces dont le nom et les méthodes sont fixés et ensuite, pour chaque plate-forme il y a du code. Ces interfaces sont accessibles par JavaScript (via la technologie XPConnect), mais pas seulement. Un composant XPCOM peut aussi être accessible à un autre langage : il existe par exemple PyXPCOM pour accéder aux composants de Mozilla depuis Python.

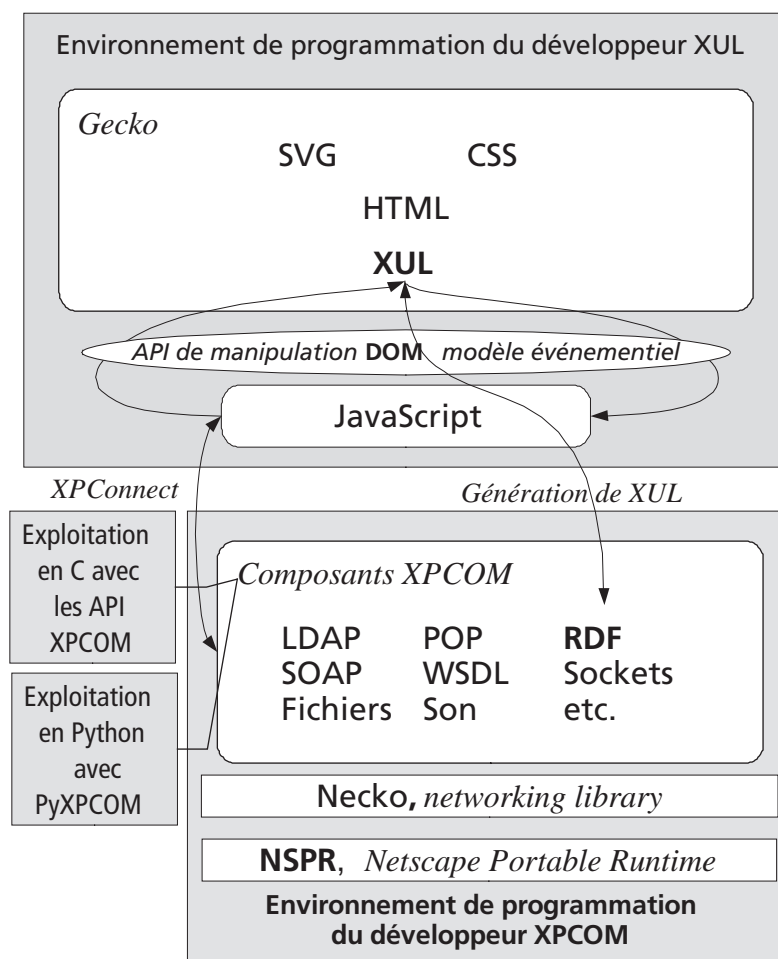


Figure 1-4
L'organisation générale du XPFE

Il existe des composants pour tout : services web, services d'annuaire LDAP, messagerie (notamment les communications avec un serveur POP), etc. Il y a même un projet de composant pour inclure la connexion à des bases de données MySQL ou PostgreSQL via XPCOM, accessible par JavaScript ! Vous pouvez bien sûr créer vos propres composants, en C++ si vous voulez être plus proche du système, ou en JavaScript si vous n'avez pas besoin de fonctions avancées.

L'ensemble du code qui concerne les protocoles réseau est intégré dans Necko, la *Networking Library* de Mozilla. En outre, le code utilise NSPR, Netscape Portable Runtime, un ensemble destiné à rendre les opérations les plus courantes (comme la gestion des entrées/sorties ou des *threads*) indépendantes de la plate forme : il s'agit donc d'une couche d'abstraction.

En résumé, sur la base du NSPR se situent les composants XPCOM, dont l'implémentation peut varier en fonction de la plate-forme, mais qui ont des interfaces normalisées. Ces composants peuvent utiliser la bibliothèque réseau Necko et sont accessibles depuis un autre langage comme JavaScript. JavaScript peut interagir à la demande de XUL avec XPCOM mais aussi avec le moteur Gecko, via le DOM. Le moteur Gecko se charge finalement d'afficher le document XUL, en prenant en compte les technologies connexes, comme CSS.

Cette introduction, peut-être un peu longue, est néanmoins très importante dans un ouvrage sur XUL. En effet, il n'est pas possible de concevoir XUL simplement comme un XML « amélioré » permettant de créer une interface graphique (GUI). XUL est une technologie au cœur d'un ensemble ; elle n'est qu'une fraction de l'immense *framework* Mozilla.

ALTERNATIVES **Client léger et client lourd**

Nous présentons les avantages de XUL. Il reste cependant quelques cas où XUL ne peut pas être la « solution miracle ».

Pour mettre en œuvre une interface web simple, sans capacité particulière, le couple HTML (avec éventuellement un peu de JavaScript) ou un langage dont le code sera exécuté par le serveur reste toujours aussi efficace, simple et rapide à implémenter.

Inversement, pour développer une application très proche du système, XUL ne sera pas non plus le mieux adapté : sa portabilité l'empêche justement de se rapprocher trop des composants fondamentaux. Il faudra alors se tourner vers d'autres langages plus éprouvés, notamment C/C++, ou bénéficier de XUL en programmant des composants XPCOM !

Mais à moins d'être dans l'une de ces situations « extrêmes », il y a aura toujours une bonne raison d'utiliser XUL ;).

Pourquoi choisir XUL ?

En effet, au-delà du côté standard, Open Source et « bonne image » de Mozilla, pourquoi finalement ne serait-il pas plus utile de développer une application graphique en utilisant par exemple C++, du code natif, ou encore un client léger agrémenté d'un peu de DHTML ? Les qualités de XUL sont pourtant déterminantes.

Un langage facile à apprendre

XUL, fondé sur XML, offre la garantie que le code sera toujours compréhensible car les différents *tags* sont parlants : *window*, *button* ou *tree* sont des éléments aisément compréhensibles, même avec seulement quelques rudiments d'anglais.

RETOUR D'EXPÉRIENCE LeMonde.fr

Dans une interview donnée au Journal du Net, le directeur informatique du site lemonde.fr indique que la migration d'une solution propriétaire sous Solaris vers une solution PHP pour le site et XUL pour la partie backend a apporté un nombre important d'améliorations au service.

Google a aussi une version XUL

► <http://www.google.com/mozilla/google.xul>

RETOUR D'EXPÉRIENCE Les nombreux cas d'utilisation de la plate-forme Mozilla

Sur le lien qui suit, vous pourrez voir les principaux projets recensés par Mozilla.org réutilisant le XPFE. Beaucoup intègrent Gecko, quelques-uns reprennent la totalité du XPFE. Lorsque vous aurez parcouru la liste, vous réaliserez que Mozilla est partout !

► <http://www.mozilla.org/university/HOF.html>

ALTERNATIVE XAML

Microsoft, dans la future version de son système d'exploitation Longhorn, propose de reprendre le concept de description d'interfaces graphiques avec XAML (prononcez : « Zammel »), une description XML qui elle aussi décrirait des interfaces mais serait intégrée directement avec WinFX (successeur de l'API Win32), au cœur même du système d'exploitation. Cependant, cette technologie n'a pas été éprouvée et ressemble fortement à XUL. Beaucoup prétendent en effet que XAML serait la version XUL propriétaire de Microsoft et, dans ce cas, XUL court le risque d'être écrasé par la puissance de l'éditeur de Redmond. C'est pourquoi, il y a différents projets d'intégration de XUL dans un environnement graphique, avec notamment son utilisation pour le projet GNOME (un des deux principaux environnements graphiques sous Linux), ou KaXUL, pour créer des fichiers natifs KDE, l'autre principal environnement graphique libre, à partir de XUL. Tous deux permettraient d'apporter une réponse à la nouvelle technologie développée par Microsoft. Pour ses défenseurs, XUL est une technologie mature qui a besoin maintenant des efforts de la communauté.

Les autres technologies comme JavaScript seront déjà familières à la plupart des lecteurs ayant travaillé en environnement web ; pour les autres, elles ne demanderont qu'un apprentissage minime. Le seul point délicat est la couche objet du JavaScript. Là où, dans un projet web, quelques lignes procédurales suffisent dans la plupart des cas, dans un projet XUL, la majeure partie de la programmation sera en JavaScript objet, pour des raisons de clarté et d'efficacité.

Portable

Les problèmes de portabilité sont complètement masqués lors du développement d'une application XUL. Il n'existe quasiment aucun code dépendant de la plate-forme, les composants XPCOM proposent les mêmes interfaces sous tous les systèmes et la plate-forme Mozilla est disponible sur un nombre impressionnant de systèmes. On peut ainsi télécharger des binaires de Firefox pour Microsoft Windows, Mac OS, Linux et autres *unices* comme AIX, OS/2, Solaris ainsi que la famille des BSD.

Sans limite !

Lorsque vous vouliez utiliser DHTML pour vos développements web, vous aviez à jongler entre les fonctions très diverses des différentes familles de navigateurs : combien de problèmes de réalisation entre le document.all d'Internet Explorer, la nécessaire compatibilité avec Netscape et ses implémentations étranges de JavaScript, la spécificité de Mozilla qui gère le modèle événementiel du DOM (contrairement à Internet Explorer) ! Tous ces problèmes appartiennent désormais au passé puisque Mozilla implémente parfaitement le support de JavaScript

ALTERNATIVE AJAX

Une alternative qui monte en ces temps d'intense communication client/serveur est AJAX : Asynchronous JavaScript And XML. Comment se structure cette méthode ? D'abord, le client, à l'aide d'un objet XMLHttpRequest (nous le verrons au chapitre sur JavaScript), envoie une requête au serveur. Le serveur répond sous forme de contenu XML. La réponse est analysée par le client et, à l'aide de JavaScript, de CSS et de HTML, ce dernier modifie dynamiquement l'interface sans recharger intégralement la page web. C'est donc une technique qui s'exécute dans le navigateur et qui fait appel à du DHTML (Dynamic HTML). C'est ainsi que fonctionne l'application de courrier de Google Gmail.

Cette solution peut sembler fort efficace, néanmoins il faut avoir conscience de ses limites :

- de très gros problèmes de compatibilité : Gmail par exemple ne propose la version dynamique que pour les navigateurs de dernière génération (comme Safari, Firefox ou Internet Explorer 6). Les navigateurs plus anciens se voient proposer une version HTML plus simple du site.

- HTML n'est pas adapté à la construction d'interfaces graphiques. Même s'il est possible de reproduire le comportement d'une interface riche, HTML est avant tout un langage de présentation et non pas un langage de description d'interfaces utilisateur.
- une complexité sous-jacente : l'utilisation intensive d'XML et les jongleries entre les différentes implémentations du DOM induisent une complexité importante. Songez, en regard, à la simplicité d'un remplissage par source RDF... que nous verrons au chapitre éponyme !

AJAX est certes prometteur mais souffre de nombreuses limitations, notamment du fait qu'il s'appuie sur les fonctions du navigateur web et sa capacité à interpréter du HTML pour construire des interfaces graphiques. Or, HTML n'a pas été conçu pour cet usage ; pour preuve, la complexité que tous rencontrent pour représenter un simple arbre hiérarchique en HTML.

et des CSS et ce surtout d'une manière maintenant uniforme. Il est en outre possible d'aller beaucoup plus loin avec XUL : les nombreuses API SOAP, WSDL, XML-RPC, LDAP ou POP en font un parfait candidat dès lors qu'il s'agit d'interagir avec d'autres systèmes dans un environnement hétérogène.

En résumé...





Nous avons vu dans ce chapitre que :

- Mozilla s'est transformé ; d'une simple réécriture, le projet est devenu une plate-forme pour l'exécution d'applications complexes
- L'ensemble des technologies permettant l'exécution d'applications s'appelle le XPFE.
- XUL est la technologie permettant de créer une interface graphique et s'utilise avec d'autres standards comme CSS, JavaScript, etc.
- Les avantages de XUL sont nombreux : portabilité, respect des standards, facilité de mise en œuvre.

Nous pouvons donc maintenant nous intéresser à l'étude de cas et commencer à organiser notre travail.

2

chapitre

Fichier		Options		?																
																				
Nouveau		Editer		Supprimer																
LES MEMBRES DU FORUM			LES MESSAGES DU FORUM																	
<div>Liste</div> <div>Infos membre</div>																				
 <div>Nom : Benoit</div> <div>Profession : Geek</div> <div>Localisation : devant pc</div> <div>Age : 30</div>			<table border="1"><thead><tr><th>Titre</th><th>Date</th><th>Auteur</th></tr></thead><tbody><tr><td><input checked="" type="checkbox"/> Sujet non déplié</td><td>27/07/05</td><td>Muriel</td></tr><tr><td><input type="checkbox"/> Sujet déplié</td><td>28/07/05</td><td>Jonathan</td></tr><tr><td> Une réponse</td><td>29/07/05</td><td>Benoit</td></tr><tr><td> Une autre</td><td>30/07/05</td><td>Stéphane</td></tr></tbody></table>			Titre	Date	Auteur	<input checked="" type="checkbox"/> Sujet non déplié	27/07/05	Muriel	<input type="checkbox"/> Sujet déplié	28/07/05	Jonathan	Une réponse	29/07/05	Benoit	Une autre	30/07/05	Stéphane
Titre	Date	Auteur																		
<input checked="" type="checkbox"/> Sujet non déplié	27/07/05	Muriel																		
<input type="checkbox"/> Sujet déplié	28/07/05	Jonathan																		
Une réponse	29/07/05	Benoit																		
Une autre	30/07/05	Stéphane																		
avaxhome.com																				
Informations sur ce que fait Javascript		Non lus: 3		Total: 50																

L'étude de cas

Quelle application choisir pour bien représenter les possibilités du XPFE et comment l'organiser ?

SOMMAIRE

- ▶ Le projet
- ▶ Les technologies mises en jeu
- ▶ Les avantages de XUL ?

MOTS-CLÉS

- ▶ Modularité
- ▶ Larges possibilités
- ▶ Professionnalisme

Les applications communautaires sont en général les plus attractives : ChatZilla, le client IRC intégré dans la suite Mozilla en est un exemple. Il est toujours plus amusant de développer une application reliant des personnes différentes, ne serait-ce que pour se sentir moins seul lors des phases de test ! Un service de liste de membres, d'annuaire communautaire serait une idée, mais serait peut-être un peu trop classique et pas assez étendu. Un forum alors ? Un forum combinerait plusieurs modules et utiliserait plusieurs technologies.

Introduction

Le but de cette application est de mettre en jeu les différentes possibilités offertes par le XPFE, tout en gardant un contenu capable de satisfaire aux différents critères d'une application professionnelle.

Le forum combinerait les éléments suivants :

- avant toute chose, la prise en charge des fonctions d'inscription ;
- l'identification des utilisateurs pour accéder au forum ;
- la consultation du contenu du forum : navigation dans les différents sujets, lecture et modification des messages ;
- la liste des membres : accès à la liste des membres du forum, accès aux informations les concernant.

Nous développerons XUL Forum sous forme d'extension Mozilla ou Firefox, pour des raisons de commodité (il y a plus de limitations quand on sort du cadre « interne » du navigateur). De ce fait, notre application reposera sur un serveur, qui centralisera toutes les demandes venant des utilisateurs ayant installé l'extension : lecture, écriture, rapatriement d'informations, etc. ; le tout soutenu par une base de données (dans notre cas, MySQL). Notre serveur sera animé par le langage PHP et nous décrirons les traitements côté serveur au fur et à mesure des chapitres.

L'application sera destinée à être lancée dans une fenêtre séparée, soit via une entrée de menu, soit en créant un raccourci spécial, utilisant l'option de la ligne de commande `-chrome`, qui « cachera » la fenêtre du navigateur pour ne faire apparaître que l'extension. Nous reverrons ceci au chapitre suivant lorsque nous aborderons les différents moyens d'accéder à l'extension, une fois la première page intégrée.

Fonctionnement

Principe de fonctionnement d'un forum

Dans la jungle des nouveaux moyens de communication, que ce soit messagerie instantanée ou blogs, on retrouve l'idée des forums. Les forums sont des espaces où les utilisateurs peuvent poster des messages, organisés le plus souvent en sujets : à chaque sujet, il est possible de répondre en laissant un message, créant ainsi une structure hiérarchique qui permet un premier tri ordonné.

Un forum contient généralement une liste des utilisateurs et restreint l'accès à certains privilèges aux seuls membres enregistrés. Certains forums se limitent à l'aspect discussion, d'autres développent davantage un aspect communautaire en proposant des services complémentaires, comme l'échange de messages directs entre membres. XUL Forum se situera entre les deux : il proposera des services pour rendre le forum plus accueillant (liste des membres dans le but de constituer une communauté) sans pour autant perdre de vue sa vocation première qui est d'échanger des messages entre inscrits.

ALTERNATIVES Les forums HTML

Il existe des dizaines de forums, souvent gratuits, programmés en PHP/MySQL, en Perl ou autre, avec une sortie HTML. Cependant l'expérience utilisateur est meilleure avec une interface riche Mozilla : au lieu d'une mise en forme « présentationnelle », on dispose d'une véritable application, avec des menus, des boutons, des zones de texte, des arbres, etc.

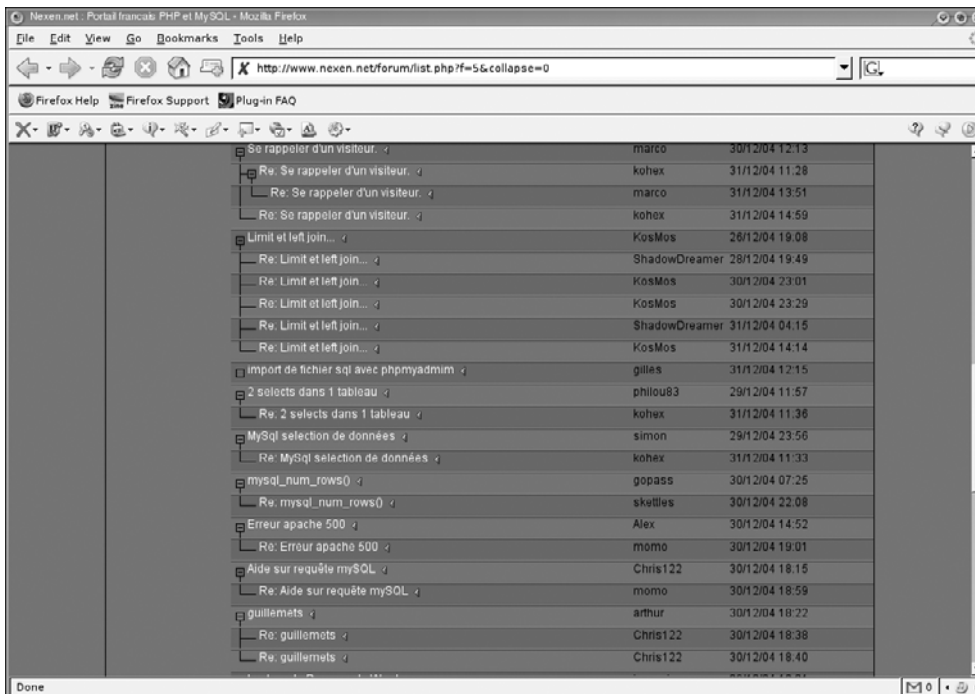


Figure 2-1
Un exemple de forum, avec la hiérarchie de messages

Le fonctionnement de XUL Forum

La vocation première de XUL Forum est de permettre des discussions entre internautes. L'utilisateur devra tout d'abord être capable de lire les messages des autres participants.

L'identification est nécessaire, notamment en ce qui concerne l'accès au service et l'envoi des messages. Il sera ainsi possible d'associer à chaque message son auteur et donc de permettre à l'utilisateur de modifier ses propres messages. De plus, la phase d'inscription préalable permettra de remplir la liste des membres du forum, liste consultable par tous les membres. Cette liste fournira des informations personnelles sur chaque inscrit : e-mail, âge, nom, etc.

Il y aura donc quatre fonctions principales : inscription, identification, lecture/modification/ajout d'un message, informations sur les membres. À chacune de ces fonctions va correspondre une partie de l'interface : c'est ce que nous allons étudier maintenant.

Les différents modules

L'inscription

POUR ALLER PLUS LOIN

Il serait bien sûr possible de recréer la page d'inscription en XUL et de l'intégrer directement à l'extension. Ce sera un très bon exercice, une fois la lecture de l'ouvrage terminée.

La première étape pour un futur membre du forum sera son inscription. Le côté communautaire du forum n'est valable que si les membres « s'impliquent » dans le forum ; c'est pourquoi nous avons choisi de ne pas laisser un anonyme poster de messages : un inconnu doit, s'il veut participer, s'inscrire !

Son inscription permettra de constituer sa fiche, qui ira remplir l'annuaire du forum. Les autres visiteurs pourront la consulter. Elle fournira son nom, sa photo, sa localisation.

L'inscription pourra se faire directement sur le serveur via une page web en PHP. Vous pourrez retrouver cette dernière sur le site du projet, car nous n'aurons pas l'occasion de détailler, dans un ouvrage consacré à XUL, les fonctions de cette page PHP !

L'identification

Une fois inscrit, l'utilisateur devra s'identifier à l'aide de son nom d'utilisateur et du mot de passe choisi lors de l'inscription. Ce sera pour lui le premier contact véritable avec XUL Forum, la première page XUL qu'il verra. Cet écran sera accessible depuis le forum à proprement parler, si jamais l'utilisateur a besoin de s'identifier sous un nom différent.

L'identification, de même que la liste des membres, seront gérées par un serveur LDAP (un service d'annuaire offrant des fonctions d'identification), avec qui nous communiquerons par l'intermédiaire de l'extension LDAP de Mozilla. Cette utilisation de LDAP est cependant facultative. Dans cet ouvrage, nous décrirons XUL Forum dans sa version avec LDAP, mais, pour l'offrir au plus grand nombre d'utilisateurs, une version sans LDAP sera fonctionnelle et disponible sur le site du projet.

B.A.-BA LDAP

LDAP est un service d'annuaire internet. Il peut être utilisé dans une entreprise pour tenir à jour une hiérarchie des employés. Il fournit aussi des mécanismes d'identification. Nous verrons tout cela en détail au chapitre 10.

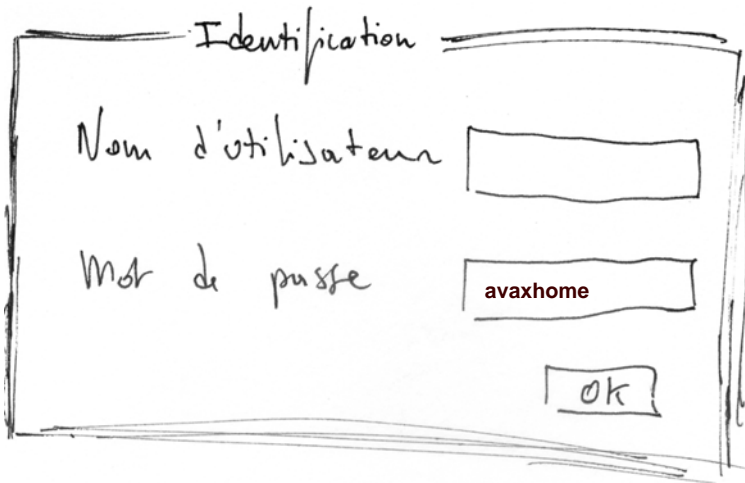


Figure 2-2

L'aspect grossier du formulaire d'identification

L'écran principal

Après s'être identifié, l'utilisateur arrive sur l'écran principal. Sa structure reprend celle de la fenêtre principale d'un navigateur : dans la partie supérieure, on retrouve les menus et les barres d'outils ; dans la partie centrale, le forum, avec la sélection des différents messages ; sur la gauche, une *sidebar* sert à donner des informations aux membres ; en bas, une barre d'état indique l'état de l'application et l'avancement d'une opération longue (lors de connexions à des serveurs distants).

L'écran principal est rempli principalement grâce à des sources de données de type RDF, pour construire de manière automatisée les principaux éléments de l'interface, comme l'arbre des messages.

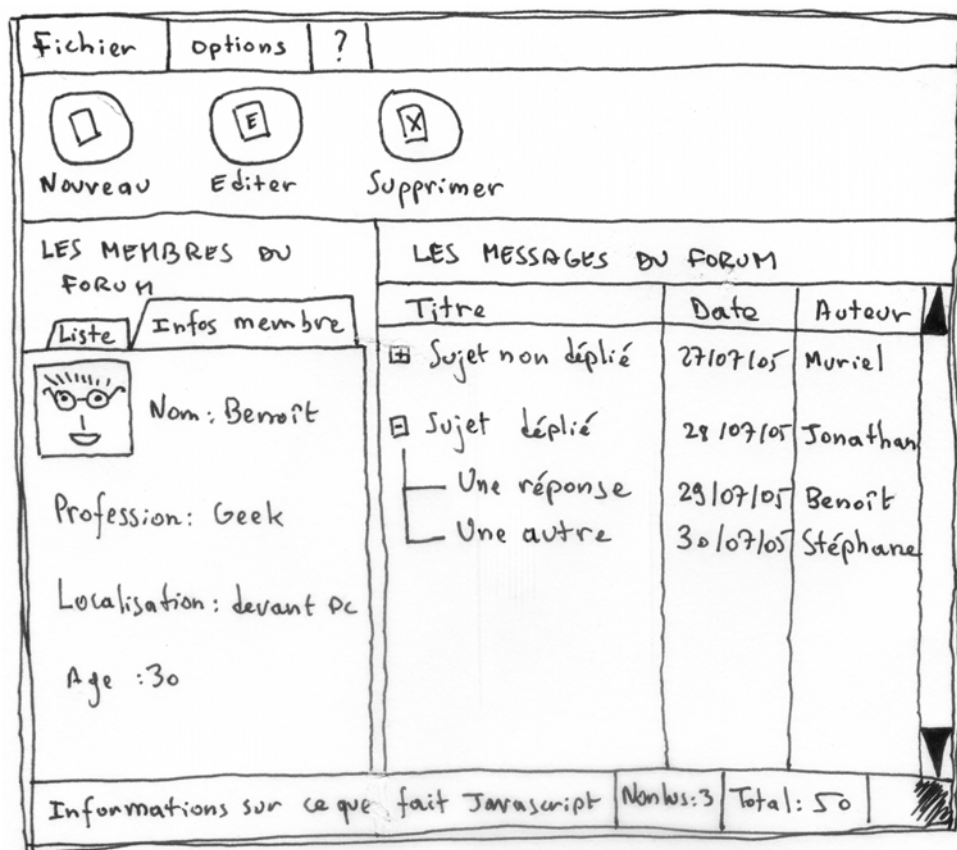
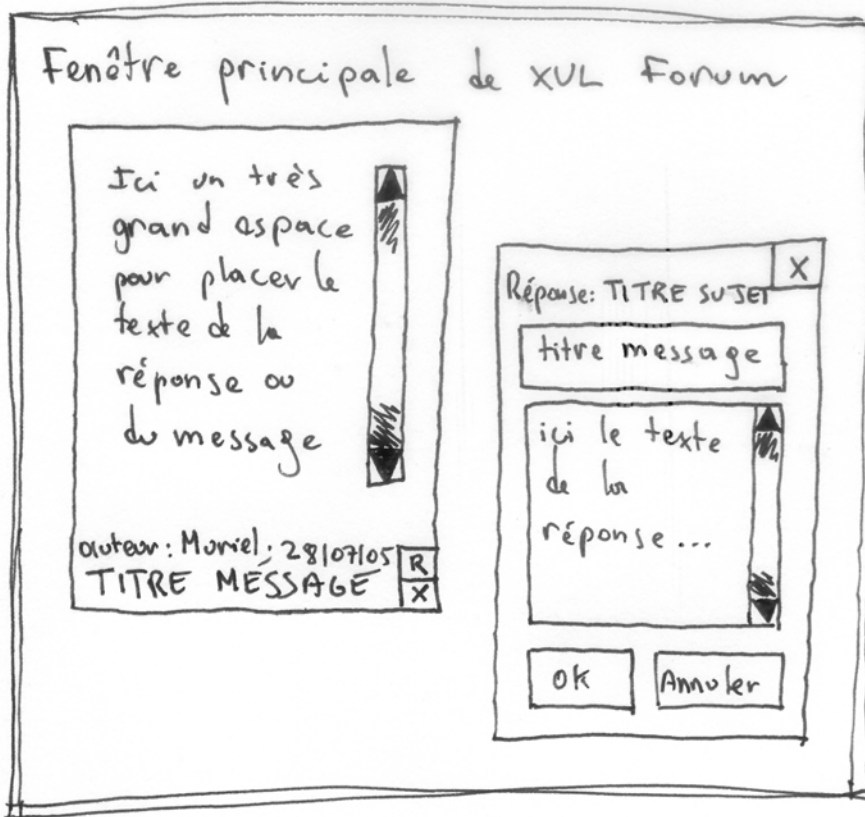


Figure 2-3
Fenêtre principale de XUL Forum

La fenêtre d'ajout/modification/lecture d'un sujet

Pour lire, modifier ou créer un sujet, nous faisons appel à une fenêtre *pop-up*, qui permet de rédiger ou de lire un sujet sans perdre des yeux la fenêtre principale du forum.

Les champs présents sont le titre et le contenu du message. L'enregistrement ou la lecture des messages se feront par l'intermédiaire d'un service web.

**Figure 2-4**

L'aspect sommaire des deux pop-ups, l'un pour répondre, l'autre pour lire un message

Les différentes parties du XPFE mises en jeu

XUL Forum nous permettra d'aborder différentes technologies offertes par l'environnement de travail que constitue le XPFE.

Les technologies « statiques »

Nous aborderons successivement :

- XUL : la structure de l'application
- CSS : mise en forme, rendu « personnalisé »
- XBL : création de nos propres *widgets*

La partie XUL permettra de nous familiariser avec ce langage : en comprendre tout d'abord les fondements au travers de quelques éléments simples, puis d'en découvrir ensuite toute la puissance grâce à des éléments en apparence plus complexes.

CSS sera l'occasion de (re)découvrir les avantages et les possibilités de ce langage, qui est toujours présent par défaut, mais que l'on peut contrôler pour définir le *look and feel* de l'application.

Enfin, XBL, eXtensible Binding Language nous permettra de créer nos propres éléments XUL, les *widgets*, et de définir leur structure mais aussi leur comportement, en créant ainsi un widget statico-dynamique (nous verrons les différents constituants d'un *widget* XBL dans le chapitre qui y est consacré).

Les technologies « dynamiques »

Nous étudierons plusieurs technologies connexes permettant de « dynamiser » notre application :

- RDF pour alimenter le remplissage de *widgets* avec une source tierce de données ;
- JavaScript, l'incontournable langage de *script* et son modèle DOM ;
- l'utilisation de JavaScript avec les services web (SOAP et WSDL notamment) ;
- XPIInstall, Cross Platform Install, pour installer et distribuer facilement notre application.

RDF permettra de construire l'arbre des messages du forum en nous aidant d'une source fournie par le serveur distant. JavaScript dynamisera l'application, à travers son Document Object Model (DOM) permettant de modifier dynamiquement l'interface XUL et commandera les opérations concernant les services web : ce seront des opérations « actives », comme la modification ou l'ajout d'un message. Enfin, après avoir « emballé » notre application, nous utiliserons XPIInstall pour la distribuer (à l'aide d'un fichier *.xpi*) et pour commander son installation depuis une page web.

Quels seront les avantages de XUL dans ce cas ?

On peut se demander l'intérêt de choisir ce type d'application dans le cadre d'un ouvrage voulant présenter XUL : en effet, il existe une multitude de solutions de forum disponibles sur Internet, en HTML et ne se limitant pas à une certaine classe de navigateurs.

Des modules réutilisables

À part le module principal qui concerne le forum, les parties de l'application telles que l'identification, l'inscription, la liste des membres seront réutilisables dans le cadre d'un autre développement. En effet, que ce soit dans un intranet ou dans une interface d'administration par exemple, l'identification est un passage obligé et lorsque vous développerez votre propre application, vous pourrez facilement réutiliser le code existant. De même, lorsque nous serons amenés à lier XUL Forum au service d'annuaire LDAP (pour la liste des membres), le fichier JavaScript correspondant pourra instantanément être réutilisé pour une autre application.

Une approche croissante des difficultés

En séparant des parties de l'application telles que l'identification de la fenêtre principale, l'approche de XUL peut se faire de manière progressive. Les éléments qui seront utilisés dans les premières pages seront familiers au développeur web : zones de texte, boutons, cases à cocher, etc. Une fois le lecteur habitué à ces quelques éléments XUL, il sera plus facile d'aborder des *widgets* spécifiques qui auraient pu être déroutants au premier abord.

Une large utilisation des possibilités du XPFE

Même si dans certains cas l'utilisation de RDF ou de XBL peut sembler trop « entendue », l'application se veut avant tout pédagogique et son interaction avec différents systèmes, ainsi que son utilisation des différentes API offertes par Mozilla, en font un projet de choix pour défendre la cause du navigateur libre face à des solutions propriétaires.

Un environnement professionnel

Enfin, en dépit du fait que XUL Forum sera aussi un démonstrateur technologique, on se retrouvera néanmoins dans une situation classique de développement : traitement des données par un serveur distant (par communication RDF et services web), intégration avec une application tierce (LDAP), prise en charge de l'internationalisation (DTD), etc., sont des problèmes qui se posent souvent lors d'un développement et pas seulement en utilisant XUL.

RETOUR D'EXPÉRIENCE

Le parti socialiste aime XUL !

► <http://www.idealx.com/press/communique.php?id=34>

Points noirs à l'utilisation de XUL

Il ne faut néanmoins pas oublier les désavantages qu'implique le choix de XUL :

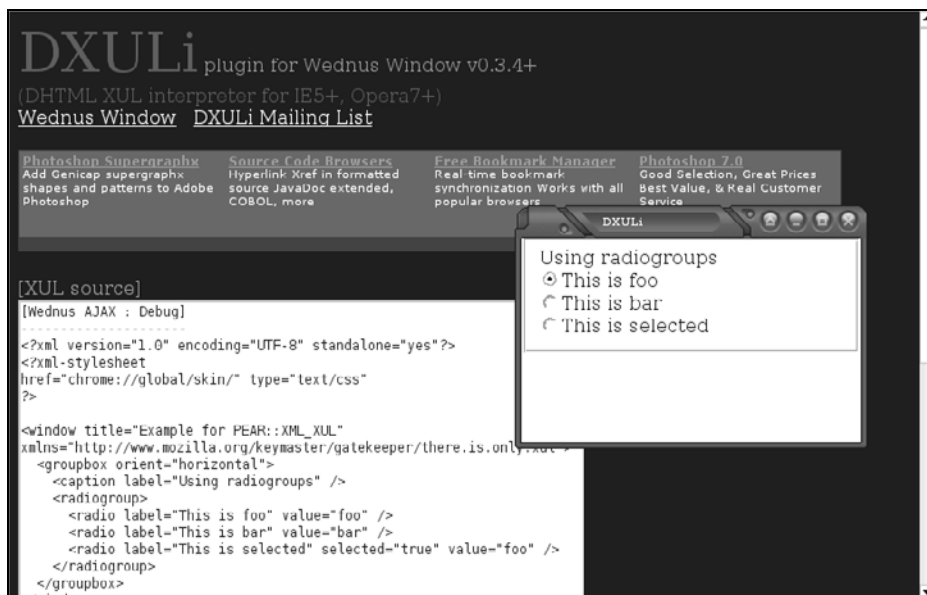
- incompatibilité avec la majorité des utilisateurs ;
- liaison (trop ?) distante avec le serveur.

Même si à l'heure de la rédaction de cet ouvrage, le pourcentage de navigateurs Mozilla dépasse les 10 % de part de marché, il reste quasiment neuf utilisateurs sur dix qui ne pourront pas profiter du forum. C'est pourquoi, si cette application était critique pour la constitution d'un site ou d'un intranet, il faudrait envisager une version HTML du forum pour ceux qui ne pourraient profiter de la version XUL. La partie serveur de XUL Forum serait ainsi constituée par le code de base du forum HTML ; simplement, au lieu d'envoyer les résultats sous forme de page web, elle pourrait les envoyer sous forme de RDF et implémenter les services web nécessaires à la version XUL.

POUR ALLER PLUS LOIN **Déploiement de XUL**

XUL commence à sortir du cadre de Mozilla et l'on peut espérer voir bientôt des interpréteurs de XUL disponibles pour d'autres plates-formes que le XPFE. Le projet cité ci-dessous utilise du DHTML (Dynamic HTML), c'est à dire CSS, JavaScript et HTML, pour interpréter du XUL et le traduire en HTML. Il fonctionne certes sur des fichiers basiques, mais surtout dans Internet Explorer, Opera et bien sûr Mozilla. À suivre !

- ▶ <http://wednus.com/wednus0.3.4/doc/samples/DXULi.htm>



Un autre aspect négatif est la « sur-utilisation » de méthodes de communication distantes avec le serveur. Dans le cas d'un forum HTML, la page est créée directement par le langage côté serveur et ne passe pas par tout le procédé d'utilisation des services web. L'implémentation côté serveur

est plus lourde que pour créer une simple page web. Mais peut-être est-ce justement là ce qui démontre la force de XUL : agir là où le HTML atteint ses limites et où le client lourd entraîne trop de complications.

En résumé...

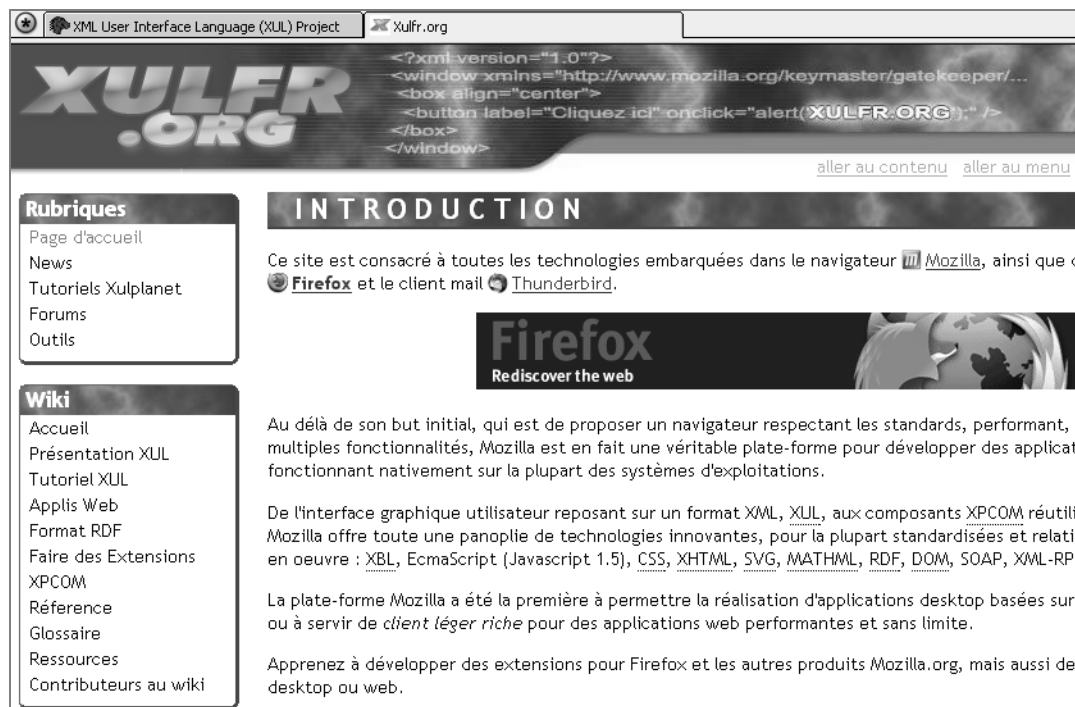
Nous avons donc vu au travers de ce chapitre :

- la structure de notre application et ses différents modules : inscription, authentification, gestion des membres, lecture/modification/création d'un message ;
- en quoi cette application couvre une grande partie des possibilités du *framework* Mozilla ;
- pourquoi XUL s'adapte bien à ce type de projet.

Il est grand temps maintenant de commencer à vraiment écrire du XUL !

3

chapitre



Premiers pas en XUL

Sortez votre éditeur favori... l'écriture de XUL peut commencer !

SOMMAIRE

- ▶ Premier fichier XUL
- ▶ Modèles de boîte
- ▶ Espaceurs

MOTS-CLÉS

- ▶ boîte verticale/horizontale
- ▶ répartition de l'espace dans une boîte
- ▶ attribut flex

OUTILS Éditeur de fichiers XML

Comme vous aurez à manipuler différents langages, le choix d'un éditeur de texte s'impose. Celui-ci doit être capable de gérer le format XML, d'enregistrer en Unicode (quasiment indispensable), de reconnaître un peu de syntaxe JavaScript et CSS, etc. Sous Microsoft Windows, il existe bon nombre d'éditeurs gérant ces langages : UltraEdit, HTMLPad, PHPEdit si vous aimez PHP..., mais aussi gvim ! L'éditeur phare du monde Unix s'utilise aussi bien sous Unix que sous Windows et propose une version graphique pour les réfractaires au mode console. Il gère l'Unicode, il est facilement configurable et sur vim.org, vous pourrez trouver des extensions pour vous faciliter la tâche d'édition du XML. Si bien sûr vous préférez Emacs, il vous est aussi possible de l'utiliser ! Le choix entre les deux éditeurs est une affaire de goût.

L'écriture de code commence concrètement avec ce chapitre : d'abord un premier fichier XUL, pour se familiariser ou se re-familiariser avec l'écriture XML et la logique de groupement en boîtes ; ensuite, une amélioration pour vraiment « penser XUL ».

Un premier fichier XUL

Pour commencer, nous allons construire l'écran d'identification. Il est composé d'un nombre limité d'éléments et constitue le candidat idéal pour un premier fichier XUL.

L'idée est d'utiliser directement, au fur et à mesure que le besoin s'en fait sentir, les éléments nécessaires à la construction de notre interface. Ainsi, dans cette page en apparence simple, nous ne nous familiariserons qu'avec des éléments faciles à appréhender. Les chapitres suivants permettront de nous attaquer au gros de l'interface et d'introduire des éléments plus complexes.

Le contenu du tout premier fichier XUL !

```
<?xml version="1.0" encoding="iso-8859-1"?> ❶
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul"> ❷
  Mon premier document XUL. ❸
  <!-- mon premier commentaire XUL -->
</window> ❹
```

On indique d'abord ❶ que le document qui va suivre est un document XML : c'est le prologue XML `<?xml...` qui se charge de ceci. Il précise également l'encodage iso-8859-1 pour l'alphabet des pays européens qui permettra d'identifier des caractères accentués. Si votre éditeur de code gère les fichiers XML au format Unicode, vous pouvez vous passer de l'attribut *encoding* (les fichiers XML sont par défaut en Unicode).

On ouvre ensuite ❷ un élément `<window>` qui sera l'élément racine. On spécifie son espace de nommage XML, en passant une URL spéciale (qui fait référence au titre du film « SOS fantômes »), qui sera reconnue directement par l'analyseur comme indiquant que ce qui va suivre sera du XUL. On tape ensuite quelques lignes de texte ❸ puis enfin on ferme l'élément racine ❹.

Enregistrez ce fichier dans votre éditeur favori, puis lancez Mozilla (ou Firefox, ou autre...) et pointez-le vers ce fichier au travers de l'*url file*. Vous pouvez par exemple taper dans la barre d'adresses `file://c:/jonathan/monexemple.xul`. Vous verrez alors s'afficher une splendide page blanche. C'est logique ! En effet l'organisation des documents XUL reflète

B.A.-BA Espace de nommage XML

Dans un document XML, l'espace de nommage (attribut `xmlns`) permet d'éviter des conflits entre différents noms d'éléments. Par exemple, imaginons que dans ce document XML l'on veuille inclure des balises HTML (il est possible de mélanger HTML et XUL). Comme XUL et HTML ont un nom d'élément en commun, par exemple `iframe` que l'on peut retrouver dans les deux langages, s'il n'y avait pas d'espace de nommage, l'analyseur ne saurait pas si l'on parle d'un `iframe` XUL ou HTML. Comme on a appliqué l'espace de nommage XUL à l'élément racine, tous les éléments fils seront des éléments XUL.

On aurait pu aussi ajouter à l'élément `window` l'attribut `xmlns:html="http://www.w3.org/1999/xhtml"` et à ce moment là, pour utiliser l'élément `iframe` HTML, on aurait écrit `<html:iframe>`. Nous verrons ceci plus en détail lors de l'intégration HTML avec XBL. Pour l'instant, l'important est de comprendre que l'espace de nommage permet de préciser à quel langage font référence les éléments (ici ce sont des éléments XUL).

l'organisation de l'interface modélisée. Il est donc normal qu'un texte ne puisse être placé indépendamment de tout, dans le vide. XUL décrit avant tout une interface et par conséquent nous aurons un élément destiné à afficher du texte : l'élément `<description>`.

Le premier fichier XUL... qui fonctionne !

```
<?xml version="1.0" encoding="iso-8859-1"?>
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  ➤ there.is.only.xul">
  <description>Mon premier document XUL.</description>
</window>
```

Maintenant, vous devriez pouvoir afficher sans problème cette page dans votre navigateur Gecko favori ! Si vous ne voyez rien s'afficher, c'est probablement que vous avez fait une erreur en recopiant.

ASTUCE XUL en dehors du navigateur

Vous pouvez également lancer ce document XUL « seul », c'est-à-dire sans ce qui compose le navigateur (menus, barres d'outils), pour donner l'impression que le fichier est une application autonome. Il faut alors passer l'option `-chrome` à la ligne de commande du navigateur. Sous *nix, on tapera :

```
firefox -chrome /home/jonathan/xul/monpremierfichier.xul
```

tandis que sous Windows, on pourra exécuter :

```
"C:\Program Files\mozilla.org\Mozilla\mozilla.exe" -chrome
  ➤ c:\jonathan\monfichier.xul
```

par exemple.

B.A.-BA Prologue XML

Le prologue XML est constitué par les premières lignes du document XML : `<?xml . . .`

Les documents XML sont prévus pour être encodés en Unicode par défaut, c'est-à-dire avec un jeu de caractères qui utilise deux octets par caractère et qui permet de classer des caractères complexes, des caractères accentués, des idéogrammes et des caractères d'autres alphabets dans un grand catalogue global.

Cependant, votre éditeur favori peut ne pas gérer l'Unicode : si vous ne voyez pas d'options mentionnant ce terme, c'est probablement que vous êtes en train d'éditer un fichier en latin-1 (jeu de caractères le plus courant pour les pays européens), qui n'utilisera qu'un seul octet par caractère.

Il faut alors préciser à Mozilla que lorsqu'il lira ce fichier, ce ne sera pas un fichier Unicode, mais un fichier encodé en latin-1 ou iso-8859-1, afin qu'il ne cherche pas de second octet par caractère là où il n'y en a pas ! Si vous vous êtes trompé dans le prologue XML, vous aurez probablement une erreur à la lecture de votre document et Mozilla vous indiquera la ligne et le numéro du caractère fautif. Enfin, dernière remarque, si vous utilisez le symbole Euro et que vous n'éditez pas un fichier Unicode, vous devrez spécifier un encodage iso-8859-15, une variante du iso-8859-1 comprenant notamment le symbole Euro.

Le lien ci-dessous vous permettra de visualiser sous une forme agréable l'ensemble des caractères Unicode, afin de vous rendre compte de leur étendue.

► <http://www.ianalbert.com/misc/zoom-unicode.php>

Le premier problème concernant notre document est son aspect. Le fond blanc, la police semblent être une faute de goût par rapport au reste du navigateur ! En effet, nous n'avons pas spécifié de feuille de style. Comment obtenir un style graphique en accord avec le reste du XUL présent dans le navigateur ? XUL propose une solution à ce problème sous la forme d'une feuille de style globale que l'on inclura généralement à nos documents et qu'il sera par la suite possible de modifier. En attendant d'aborder le chapitre sur les CSS, nous nous contenterons d'ajouter :

```
<?xml-stylesheet href="chrome://global/skin" type="text/css" ?>
```

en début de notre document pour qu'il prenne en compte le style général du navigateur. Vous pouvez si vous le désirez voir à quoi ressemble cette feuille de style, en pointant votre navigateur à l'adresse `chrome://global/skin`. Après cette petite modification, le style d'ensemble est déjà plus agréable à l'œil : la couleur de fond, la police sont en accord avec le reste du navigateur.

Outils Interpréteur de XUL

Le choix de l'application est arbitraire ; toute application implémentant Gecko est en théorie capable d'interpréter un document XUL : on peut utiliser Firefox, Thunderbird ou Mozilla pour lire ce fichier. Cependant, en pratique, il est en général beaucoup plus simple de tester votre XUL dans le cadre du navigateur, tant pour les « bonus » offerts au développeur (par exemple, le *debugger* JavaScript, la mise en relief des éléments de bloc, etc.) que pour les facilités telles que le bouton recharger ou la barre d'adresse, ou encore même les onglets qui permettent de jongler entre les différents documents XUL composant notre application et aussi surtout parce que votre application sera lancée dans le navigateur !

Un projet actuellement en cours de développement est XULRunner, un environnement d'exécution pour les applications XUL, comprenant la majeure partie du XPFE mais n'intégrant pas les composants inutiles, comme un navigateur. Il permettrait aux gens désireux d'utiliser XUL de ne pas avoir à utiliser Firefox ou Mozilla. Son utilisation est décrite en annexe A.

En attendant, le plus sage est d'utiliser soit Mozilla, soit Firefox pour développer.

Alternatives Autres interpréteurs de XUL

Gecko n'est pas le seul moteur capable d'interpréter un fichier XML pour en tirer une interface graphique. Il existe quelques projets parallèles, en Java par exemple, permettant d'interpréter un document XML et de le transformer en une représentation Swing/AWT (les *toolkits* graphiques en Java). L'idée est de reprendre le concept de XML pour décrire l'interface utilisateur et de l'interpréter au moment de l'exécution pour construire l'interface. Mais ils ne sont pas compatibles pour autant !

<http://xul.sourceforge.net> vous permettra d'aller plus loin dans ces autres « XML user interface languages ».

Il ne faut surtout pas confondre XUL et XML-UI. XUL est la dénomination réservée au langage utilisé au sein de Mozilla et XML-UI désigne les autres langages fondés sur le même principe. L'utilisation du terme XUL pour désigner d'autres langages XML-UI serait, selon la communauté Mozilla, abusif. Ainsi, sur le site précédemment cité, pensez bien que ce n'est pas le XUL de Mozilla dont il est question et que pour un développeur de Mozilla, l'auteur devrait utiliser le terme XML-UI !

À la découverte des premiers éléments...

Le premier exemple que nous avons vu n'est pas très intéressant. En fait, il vous permet juste de voir que votre navigateur est capable d'afficher un document XUL minimaliste et capable de s'intégrer dans la charte graphique du navigateur. Nous allons maintenant pouvoir nous intéresser à quelque chose de plus complet en écrivant l'écran d'authentification.

Première version de l'écran d'authentification

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="chrome://global/skin" type="text/css" ?>

<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul" >
  <description value="Merci de vous identifier à XUL Forum" /> ❶
  <description value="Votre nom : " /> ❷
  <textbox id="xf-ident_nom" /> ❸
  <description value="Votre mot de passe : " /> ❹
  <textbox id="xf-ident_pass" type="password" value="xxx" /> ❺
</window>
```

ATTENTION Attributs ID

Une précaution consiste à faire débiter tous les attributs ID de nos éléments XUL par xf (comme XUL Forum). Ceci permet d'éviter des erreurs, comme appeler une barre de statut « statusbar » alors qu'un élément possède déjà cet ID (c'est la barre de statut du navigateur). Il est sage de spécifier des ID sur la plupart des éléments susceptibles d'être manipulés dynamiquement. Ceci permettra par la suite, lorsque nous programmerons la partie en JavaScript de XUL Forum, d'accéder à ces éléments rapidement, grâce à leur identifiant.

Ceci est la première version que l'on aurait tendance à écrire pour effectuer un rendu du document. D'abord une petite ligne d'introduction ❶, sous la forme d'une variante de l'élément `<description>`, ensuite des champs texte ❸ et ❺ précédés par un texte ❷ et ❹ indiquant leur fonction. L'attribut `type` permet de faire apparaître de petites étoiles dans le deuxième champ texte ❺.

Si vous essayez cet exemple dans votre navigateur, vous remarquerez deux problèmes au niveau du placement des éléments. Tout d'abord, les éléments `description` et les zones de texte sont placés les uns en-dessous des autres. D'autre part, les zones de texte prennent toute la largeur possible, c'est-à-dire, dans le cas d'un lancement dans le navigateur, toute la largeur de l'écran, ce qui est plutôt disgracieux.

Corriger les premières erreurs : utilisation de boîtes

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="chrome://global/skin" type="text/css" ?>

<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul">
  <description value="Merci de vous identifier à XUL Forum" />
  <hbox> ❶
    <description value="Votre nom : " />
    <textbox id="xf-ident_nom" />
  </hbox>
  <hbox> ❷
    <description value="Votre mot de passe : " />
    <textbox id="xf-ident_pass" type="password" />
  </hbox>
  <hbox> ❸
    <button label="OK" />
  </hbox>
</window>
```

ALTERNATIVE Tag <box>

Comme vous l'aurez deviné, `<vbox>` représente une boîte verticale et `<hbox>` une boîte horizontale. Ce sont en fait des raccourcis pour : `<box orient="vertical">` et `<box orient="horizontal">`. Vous pouvez utiliser l'élément qui vous plaît le plus.

Cet exemple est plus adapté pour obtenir un résultat satisfaisant. Pour ce faire, on a placé tous les éléments censés être sur la même ligne dans des boîtes horizontales ❶ ❷ et ❸ et on a empilé les boîtes.

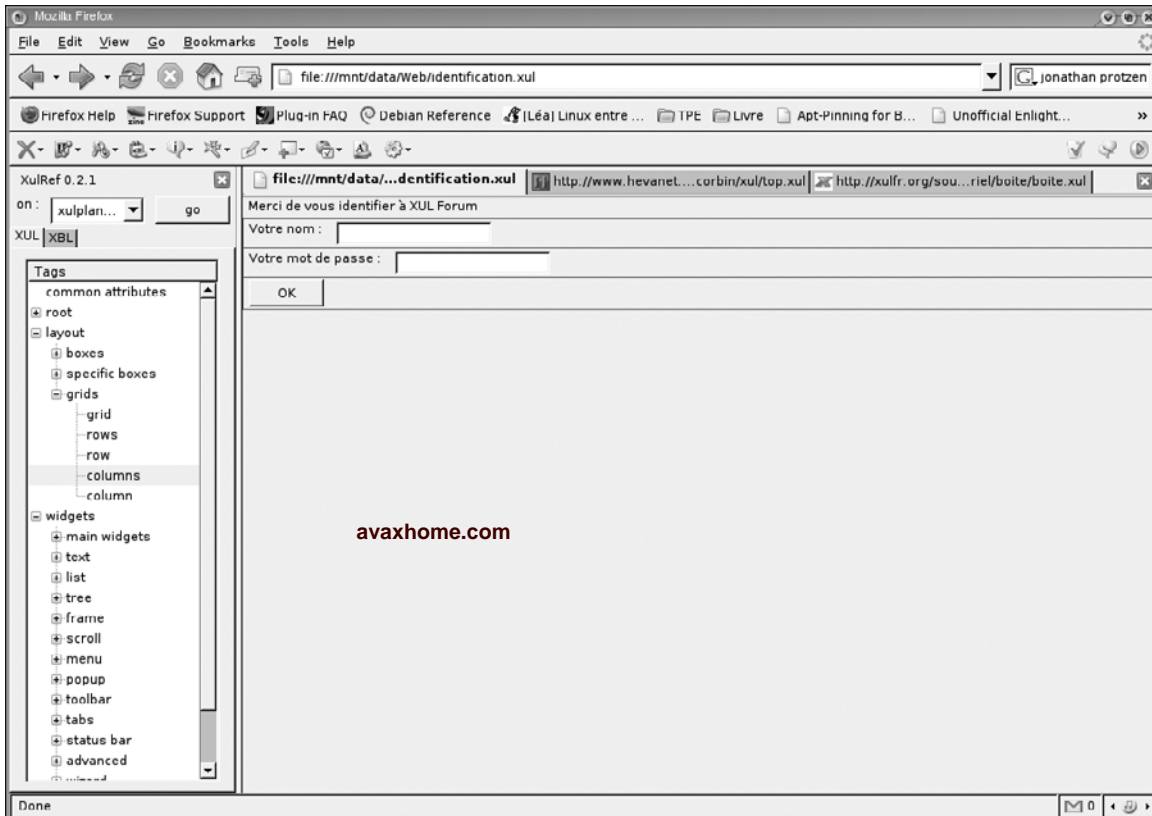


Figure 3-1 Mise en relief des boîtes horizontales dans notre premier exemple

Il faut en effet comprendre le modèle de boîte de XUL. Contrairement à certains environnements de développement qui permettent de placer les éléments au pixel près, XUL, comme la librairie GTK d'ailleurs, place ses *widgets* dans des boîtes. La fenêtre (l'élément `<window>`) se comporte comme une boîte verticale. Chaque élément est placé en-dessous du précédent et prend toute la largeur possible. Ainsi, les zones de texte et les descriptions sont placées en étages et prennent toute la largeur qui leur est offerte (dans notre premier exemple). À l'inverse, dans une boîte horizontale, les éléments prennent toute la hauteur possible et s'entassent de gauche à droite. Ainsi, dans notre second exemple, ce sont les boîtes horizontales qui s'empilent les unes sur les autres et, dans chacune de ces boîtes, le texte et la zone de texte vont le plus à gauche possible.

ASTUCE Orientation de l'élément racine

Vous pouvez définir l'orientation de l'élément `window` en utilisant l'attribut `orient`, par exemple :

```
<window orient="horizontal">
```

ASTUCE xulfr.org et modèle de boîte

► <http://xulfr.org/sources/tutoriel/boite/boite.xul>
permet de tester en ligne le modèle de boîte...
À essayer sans tarder !

Outils **DOM Inspector**

Indispensable lorsque l'on débute, l'inspecteur DOM (*Tools, DOM Inspector* ou *Outils, Inspecteur DOM*) vous permettra de naviguer dans la hiérarchie des différents fichiers, de mettre en relief tel ou tel élément (cliquez sur un élément XUL et ses contours clignoteront en rouge) et surtout, lorsque nous les aurons introduits, de voir le résultat des pages avec *overlay*.
Il s'installe avec Mozilla, en choisissant dans l'installateur les options avancées et en cochant la case *inspecteur DOM*.

Cette solution n'est peut-être pas très élégante, mais elle permet d'aligner descriptions et zones de texte sans pour autant disproportionner les éléments.

Un élément plus adapté : un tableau

Tous ces éléments vus jusqu'ici sont assez familiers au développeur web ; c'est pourquoi on aurait tendance à vouloir insérer un tableau pour garantir l'alignement de la fenêtre d'authentification. Ceci est bien sûr possible, en utilisant l'élément `<grid>` et ses sous-éléments `<rows>`, `<row>`, `<columns>`, `<column>`. Cet élément est plus adapté que les boîtes dans lesquelles on se perd vite et permet de garantir une structure nette sans « bricolage » avec les boîtes pour obtenir le résultat attendu.

On retrouve le traditionnel prologue XML.

Ainsi que l'habituel élément racine.

Un `groupbox` est l'équivalent du `<fieldset>` du HTML : il entoure ses éléments fils d'une bordure, parfois arrondie suivant le thème du navigateur.

Le `caption` est l'équivalent du `<legend>` du HTML : il définit le titre du `<groupbox>`.

C'est le tableau qui est ouvert : *grid* en anglais.

Les colonnes du tableau : ce sont des colonnes structurelles, qui ne correspondent à aucun affichage réel.

Les deux colonnes, chacune avec un attribut ID.

Maintenant, les différentes lignes du tableau.

Une première ligne.

```
<?xml version="1.0" ?>
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul">
  <groupbox>

    <caption label="Identification requise" />

    <grid>
      <columns>

        <column id="textes" />
        <column id="champs" />
      </columns>

      <rows>

        <row>
```

```

        <description value="Votre nom : " />
        <textbox id="xf-ident_nom" />
    </row>

    <row>
        <description value="Votre mot de passe : " />
        <textbox id="xf-ident_pass" type="password" />
    </row>
</rows>
</grid>
<hbox>
    <button label="OK" />
</hbox>
</groupbox>

```

◀ Chaque ligne doit posséder deux éléments car on a annoncé deux colonnes.

◀ Une autre ligne.

◀ Toujours deux éléments.

◀ La boîte permet d'éviter que le bouton ne prenne toute la largeur de la page.

Notre exemple est maintenant revu et mêle des boîtes et un tableau. Le tableau ne peut pas servir, comme dans du mauvais HTML, à améliorer l'aspect graphique de la page, en plaçant des éléments au pixel près, en faisant des bordures, etc. Il est uniquement destiné à structurer le document et permet d'aligner parfaitement les `<textbox>` et les `<description>`.

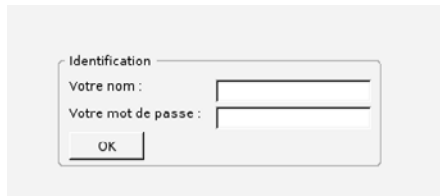


Figure 3-2 Le cadre d'identification

Finalement nous avons vu l'utilisation de quelques éléments simples : boutons, zones de texte, descriptions. Nous avons également appréhendé le modèle de boîte de XUL, en utilisant d'abord beaucoup de boîtes, puis nous nous sommes aperçus que ce n'était pas forcément la solution la plus efficace et qu'une bonne utilisation d'un tableau permettait de structurer sans problème l'interface. Une dernière touche reste cependant à apporter : il faudrait pouvoir centrer ce cadre pour avoir un rendu plus agréable.

La touche finale : spacers

Nous allons finaliser ce fichier XUL en utilisant des *spacers*. Un spacer se charge d'occuper tout l'espace possible dans une boîte. Ainsi, notre cadre `<groupbox>` sera placé dans une boîte horizontale avec à sa gauche et à sa droite deux `<spacer>`, qui se répartiront de manière équitable l'espace restant et centreront le `<groupbox>`.

```

<window>
  <hbox>
    <spacer flex="1" />
    <groupbox>
      (...)
    </groupbox>
    <spacer flex="1" />
  </hbox>
</window>

```

L'élément `spacer` admet un attribut `flex`, comme d'ailleurs la plupart des éléments XUL. Il permet de dimensionner proportionnellement les éléments d'une même boîte. En fait, dans le calcul des dimensions dans la boîte horizontale, on laisse d'abord le `groupbox` (qui n'a pas d'attribut `flex`) prendre la taille qui lui est nécessaire. Ensuite, l'espace restant est réparti entre les éléments qui, eux, possèdent un attribut `flex`. Il y a deux « unités » de `flex` au total, ce qui fait pour le premier élément la moitié de l'espace restant et pour le second, l'autre moitié. Si les attributs `flex` avaient été de deux et de un, on aurait eu deux tiers de l'espace pour la boîte de gauche et un tiers pour la boîte de droite.

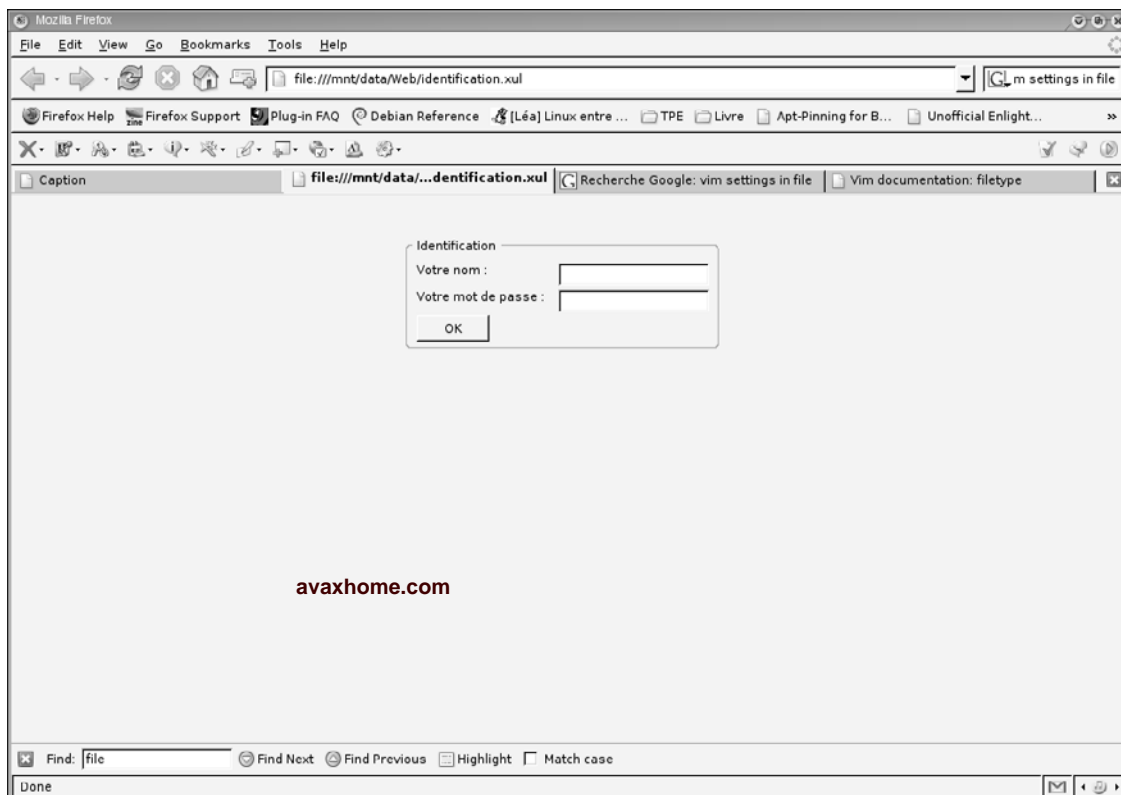


Figure 3-3 Le rendu final de la page d'identification

Si l'élément `groupbox` avait eu l'attribut `flex="1"`, la répartition en largeur d'écran aurait été : un tiers de l'écran pour le `groupbox`, un tiers de vide à droite et, enfin, un tiers de vide à gauche. Vous pouvez tester tous ces attributs sur la zone de test du site xulfr.org mentionnée plus haut, ou en modifiant vous-même le fichier de l'écran d'authentification.

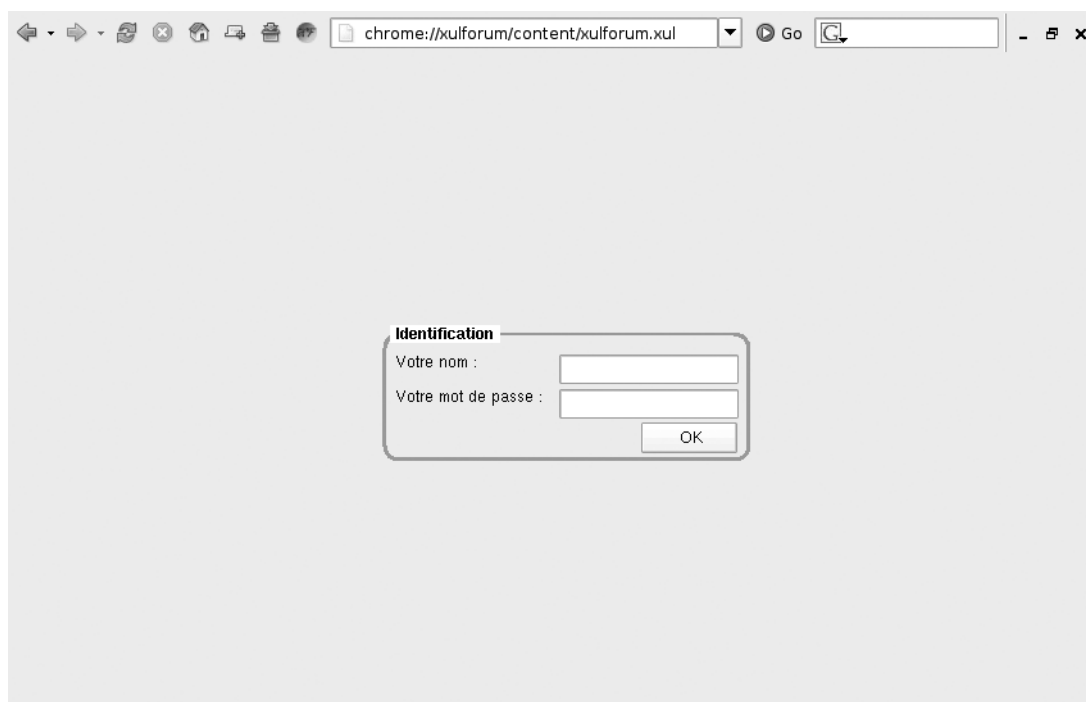
Avec cette petite modification, notre fichier est maintenant fin prêt : le cadre regroupant les éléments est centré en largeur et les textes sont parfaitement alignés. Le bouton *OK* occupe l'espace qui lui est réservé en se plaçant à gauche du cadre.

En résumé...

Nous avons vraiment dans ce chapitre fait connaissance avec XUL : ses éléments, sa conception de l'interface (du texte ne se trouve jamais seul !), sa logique de boîte, etc.

Nous pouvons maintenant nous intéresser à la place et au rangement des fichiers au sein du projet. Si vous avez envie de vérifier vos connaissances sur le modèle de boîte, vous pouvez essayer de centrer le cadre en hauteur, d'éliminer une boîte en jouant sur l'attribut `orient` de l'élément racine `<window>`, ou même de placer le bouton *OK* dans la partie droite du cadre.

chapitre 4



Une véritable extension Mozilla

Nous sommes maintenant suffisamment aguerris aux techniques XUL pour intégrer l'extension directement dans Mozilla. Bienvenue dans le navigateur !

SOMMAIRE

- ▶ Création des dossiers et premier fichier contents.rdf
- ▶ Régénération de la liste interne des extensions
- ▶ Internationalisation intégrée

MOTS-CLÉS

- ▶ DTD
- ▶ Fichier chrome.rdf
- ▶ Dossiers *content*, *locale*, *style*

ATTENTION Firefox 1.5 et Thunderbird 1.5

Si vous utilisez l'un des deux nouveaux logiciels sus-cités pour développer, des modifications ont eu lieu, relatives au processus d'enregistrement des extensions (lors du développement). Il ne faut pas éditer de fichiers `contents.rdf` ni `installed-chrome.txt`, mais seulement rajouter les fichiers `.manifest` décrits en annexe dans le dossier *chrome* de Firefox ou de Thunderbird, puis redémarrer le logiciel et c'est bon ! (le reste du chapitre s'applique toujours) Si par contre vous utilisez toujours la suite Mozilla ou son digne successeur SeaMonkey, ce chapitre s'applique dans son intégralité.

Maintenant que notre premier fichier, point d'entrée de notre application, est prêt, nous pouvons organiser la répartition des fichiers dans l'application. Comme nous avons décidé dès le départ de faire de XUL Forum une extension Mozilla, il nous faut maintenant préparer l'intégration de ce fichier et des suivants dans Mozilla. Cette étape est un peu fastidieuse mais essentielle avant de pouvoir continuer à coder. Le programmeur XUL doit être polyvalent et jongler entre différents langages !

La séparation générale des fichiers

Une extension répartit typiquement ses fichiers dans trois dossiers : *content*, *locale* et *skin*. Le premier contient les fichiers XUL, JavaScript et XBL. Le deuxième les éléments dépendants de la langue et enfin, le dernier, toutes les CSS. Pour l'instant, nous ne remplirons que le dossier *content*.

Cette séparation naturelle est redoutablement efficace. En effet, c'est en quelque sort un *design pattern* prêt à l'emploi qui est appliqué : d'un côté se trouve l'interface, de l'autre l'apparence ; enfin, l'internationalisation est bien séparée du reste. De plus, il sera possible d'enrichir l'extension : un traducteur peut préparer un dossier correspondant à sa langue, prêt à être intégré dans la version finale de XUL Forum, sans pour autant empiéter sur le travail du développeur codant en parallèle. Ce fichier de langue pourra même être installé séparément : c'est justement le cas pour les fichiers de langue française pour Firefox, qui peuvent se télécharger à part, comme sur <http://frenchmozilla.sf.net>.

Les différents dossiers de XUL Forum

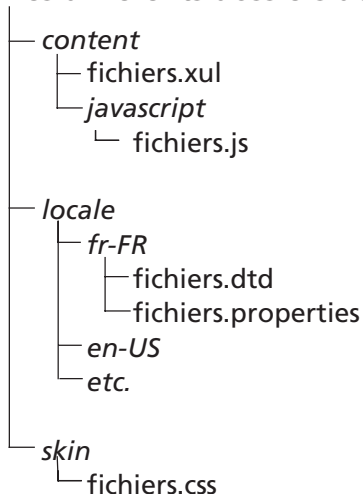


Figure 4-1

L'organisation des fichiers de XUL Forum

Contents.rdf et dossier content

Pour continuer, créez tout d'abord un dossier sur le disque destiné à accueillir votre application. Deux solutions s'offrent alors à vous :

- Vous désirez placer tous vos fichiers au cœur même de l'installation de Firefox ou de Mozilla : vous créerez alors un dossier `xulforum` dans le dossier `chrome` de l'installation de votre navigateur, par exemple `/home/jonathan/apps/firefox/chrome` ou `C:\Program Files\mozilla.org\Mozilla\chrome` pour une version Windows.
- Vous pensez changer de version du navigateur, vous avez à partager la ressource entre deux navigateurs différents, vous avez besoin de laisser l'extension sur un disque commun à deux OS : vous devez alors placer votre application dans un dossier de votre choix, mais séparé.

La différence entre les deux méthodes sera abordée plus loin dans ce chapitre, lorsqu'il faudra indiquer au navigateur où sont placés les fichiers d'extension. En fait, en plaçant votre dossier `xulforum` dans le dossier `chrome` du navigateur, vous considérez l'application comme une extension qui aurait été livrée avec le navigateur. La différence est minime et cela n'influe en rien sur le développement ; c'est en quelque sorte une habitude de travailler dans le dossier `chrome`. Libre à vous de vous y conformer !

Ensuite, il faut créer les trois dossiers mentionnés ci-dessus au sein du dossier `xulforum`, dossier racine de l'application : `content`, `locale` et `skin`. Là encore, le nom des dossiers est une sorte de convention, qu'il vaut mieux respecter pour des raisons de clarté. Nous allons maintenant créer un fichier spécial qui, une fois placé dans le dossier `content`, décrira son contenu et la façon dont Mozilla devra gérer les différents fichiers.

Ce fichier s'appelle `contents.rdf`. Nous en créerons deux autres pour les dossiers `skin` et `locale` le moment venu.

Le premier fichier RDF, `contents.rdf`, pour enregistrer le contenu de l'extension auprès du navigateur

```
<?xml version="1.0" ?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
  <RDF:Seq about="urn:mozilla:package:root"> ①
    <RDF:li resource="urn:mozilla:package:xulforum" /> ②
  </RDF:Seq>
  <RDF:Description about="urn:mozilla:package:xulforum" ③
    chrome:displayName="XUL Forum"
    chrome:author="Jonathan Protzenko"
    chrome:name="xulforum">
  </RDF:Description>
</RDF:RDF>
```


Le format de fichier RDF, déjà mentionné plus haut, fait une première apparition ici. Ce fichier `contents.rdf` décrit le contenu du dossier dans lequel il est placé. La plupart des fichiers RDF suivent cette structure : d'abord un élément `RDF:Seq` (*Seq* comme séquence) qui comprend des sous-éléments `RDF:li` (comme *list item*) et ensuite un élément `RDF:Description` pour chaque `RDF:li` mentionné plus haut.

Si l'on s'intéresse de plus près au fichier, on voit qu'il y a deux espaces de noms utilisés : un pour RDF (le format qui garantit la structure du document), un autre pour chrome, qui permet de renseigner des champs spécifiques au navigateur, champs comme l'auteur, le nom de l'application, etc., qui pourront être affichés une fois que l'extension sera installée. En effet RDF, comme nous le verrons plus tard, peut être utilisé de manière différente, sans l'espace de noms « chrome ».

On a donc un élément `RDF:Seq` ❶ qui indique que c'est un paquet que l'on décrit (attribut `about`). Il possède une entrée `RDF:li` ❷ qui contient le nom du paquet contenu dans ce dossier. Il est préférable d'utiliser un nom de paquet en minuscules et de le garder pour tous les différents champs : nom du dossier racine, nom chrome de l'extension, etc. Il n'y a qu'un seul paquet, `xulforum`, donc un seul élément `RDF:li`. L'attribut `resource` sert à indiquer le nom du paquet. Il sera toujours sous la forme `urn:mozilla:package:nomdupaquet`. On rencontrera plus loin des attributs sous la forme `urn:mozilla:locale:nomdelocale` (pour les fichiers de langue) par exemple.

Ensuite vient l'élément `RDF:Description` ❸ faisant référence à l'élément `RDF:li` : l'attribut `about` est le même que l'attribut `resource` utilisé plus haut. Il contient des attributs utilisés par le navigateur pour afficher un résumé du paquet dans le gestionnaire d'extensions (Firefox) ou dans les options (Mozilla) : nom, auteur, nom utilisé à l'affichage, etc.

Les autres attributs que l'on peut ajouter à l'élément `RDF:Description` sont :

- `authorURL`, pour indiquer l'adresse du site de l'auteur (comme son nom l'indique !);
- `description`, pour donner une description rapide de l'application ;
- `extension`, si la valeur est *true*, alors ce paquet sera considéré comme une extension et apparaîtra dans le gestionnaire d'extensions.

Nous verrons par la suite qu'il est possible d'utiliser d'autres éléments `RDF:Seq` dans ce même fichier `contents.rdf`, dans le cas des *overlays* notamment.

Comme le fichier `identification.xul` sera le « point d'entrée » de l'application, il faut le copier dans le dossier `content`, en lui attribuant un nom particulier qui en fera la page d'accueil de l'extension. On copiera

donc `identification.xul` en le renommant `xulforum.xul` (toujours dans le dossier `content`) ; comme ce fichier porte le nom du paquet, c'est celui-là qui sera affiché par défaut quand nous accèderons à l'extension.

Modification du fichier `chrome.rdf`

Maintenant que la structure de base de l'extension est prête, nous allons enregistrer cette application auprès du navigateur. Que ce soit dans le cas de Firefox ou de Mozilla, la structure interne reste la même. Tous les composants (navigateur, fichiers de langue, inspecteur DOM, *debugger* JavaScript) sont enregistrés auprès du navigateur par l'intermédiaire d'un fichier spécial, appelé `chrome.rdf`. Il est situé dans le dossier `chrome` du répertoire d'installation de votre navigateur ; ce fichier est, comme son nom l'indique, lui aussi un fichier RDF, ce qui rend son édition manuelle malaisée. C'est pourquoi nous ne le modifierons pas directement. Nous allons éditer un autre fichier, `installed-chrome.txt`, situé dans le même dossier. Ce fichier est lu par le navigateur et sert à créer le fichier `chrome.rdf`. Sa syntaxe est extrêmement simple et nous rajouterons simplement une ligne à la fin :

```
...
skin,install,url,jar:resource:/chrome/classic.jar!/skin/
classic/browser/ ❶
content,install,url,file:///mnt/data/Livre/xulforum/content/ ❷
```

La ligne ❶ est l'une des nombreuses lignes déjà présentes dans le fichier. Ici, c'est une référence à l'habillage par défaut du navigateur. La ligne ❷ est la ligne que nous ajoutons. Une ligne de ce fichier se structure en quatre éléments : le type de contenu, le mot `install`, le mot `url` et l'adresse du fichier. La plupart des éléments mentionnés dans le fichier sont compressés dans des fichiers portant une extension `.jar`, c'est pourquoi leur dernier champ diffère un peu du nôtre. Nous indiquons simplement le chemin vers notre extension *via* une URL de type `file://`.

Si vous avez choisi de placer les fichiers de votre application dans le dossier `chrome` de Mozilla, au lieu d'une *url file*, vous indiquerez une URL de type `resource:/chrome/xulforum/content/` ; *resource* indique que vous êtes dans le dossier `chrome` interne du navigateur. Il est parfois indiqué de laisser une ligne blanche à la fin du fichier, mais il semblerait que la création du fichier se fasse sans cela.

Vous pouvez maintenant supprimer le fichier `chrome.rdf` afin qu'il soit recréé à la prochaine ouverture du navigateur. Si vous utilisez Windows et Mozilla, pensez à désactiver la fonction `quicklaunch` qui vous réduit le navigateur dans la barre des tâches, sinon le fichier ne sera pas recréé.

ATTENTION Versions Microsoft Windows

Pour les versions Microsoft Windows, il est recommandé d'utiliser la syntaxe suivante :

```
content,install,url,file:///d:/livre/
xulforum/content/
```

N'oubliez pas le *slash* final ! Son absence entraînerait irrémédiablement une erreur. L'utilisation d'une barre verticale à la place des deux points est cependant facultative.

ASTUCE Regarder comment font les « vraies extensions »

Si vous voulez étudier d'un peu plus près une extension livrée avec votre navigateur, il vous suffit de copier le fichier `.jar` dans un dossier de votre choix et de le décompresser. Sous Unix, ceci se fait avec la commande `unzip`. Sous Windows, des utilitaires tels que WinZip ou WinRAR marcheront également. La structure de ces fichiers est, à peu de choses près, la même que la nôtre : dossiers *content*, *locale* et *style*. Simplement, leurs dossiers *content* sont séparés en plusieurs sous-dossiers, car ce sont de gros paquets qui nécessitent une organisation plus structurée que la nôtre. Les fichiers `contents.rdf` sont eux aussi très similaires au nôtre.

Voici le récapitulatif des étapes :

- création de la structure de l'extension en trois dossiers ;
- ajout d'un fichier `contents.rdf` destiné au navigateur ;
- placement du fichier par défaut `xulforum.xul` ;
- enregistrement auprès du navigateur via `installed-chrome.txt` ;
- régénération du fichier `chrome.rdf` après sa suppression et redémarrage du navigateur.

Si vous n'avez oublié aucune étape, vous pouvez relancer votre navigateur et le pointer à l'adresse `chrome://xulforum/content/`. Vous devriez maintenant voir apparaître l'écran d'authentification.

ATTENTION Mise à jour des extensions

Lorsque vous développez une extension, si celle-ci est intégrée au navigateur, l'interface XUL ne sera pas modifiée après chaque édition du fichier. Elle le sera après chaque redémarrage du navigateur. De même, les entités XML que nous verrons un peu plus loin ne se mettent à jour qu'après une modification du fichier les contenant et un redémarrage du navigateur. Dans tous les cas, si vous ne voyez pas de modification immédiate, pensez à vérifier que vous avez bien redémarré Firefox ou Mozilla !

Vous pourriez vous demander pourquoi l'on ne continue pas à visualiser les fichiers via l'URL `file://`. En fait, dès que nous aurons intégré l'internationalisation, nous ne ferons plus référence aux fichiers de XUL Forum que via l'URL interne `chrome://`, qui implique d'être dans le cadre du navigateur et non pas dans une URL `file://`.

Si vous voulez gagner du temps, ajouter un favori dans la barre d'outils est une solution. Vous n'aurez plus qu'à le faire pointer vers l'URL de l'extension et vous y accéderez en un clic !

Si vous ne voyez rien apparaître, essayez de vérifier les points suivants :

- tout d'abord, votre fichier `contents.rdf` est peut-être mal formé. Essayez de l'afficher dans le navigateur via l'URL `chrome` ; celui-ci vous signalera une erreur éventuelle (message `Parsing error at line X`) ;

- essayez ensuite de rechercher « xulforum » dans le fichier `chrome.rdf` ; si vous ne trouvez rien, la création du fichier a échoué et il faut trouver la source de cette erreur : faute de frappe dans le fichier `installed-chrome.txt` ? erreur dans le chemin vers l'extension ? caractère *slash* (/) final manquant ?
- avez vous bien noté le nom du fichier par défaut (`xulforum.xul`) dans le dossier `content` ?
- votre système d'exploitation ne cache-t-il pas un processus qui n'aurait pas été fermé ? Pour vous en assurer, tapez `ps aux |grep firefox-bin` ou `mozilla` dans une console Unix, sous Windows, la séquence `control-alt-suppr` fait apparaître une console qui liste les processus actifs.

Si vous n'arrivez toujours pas à faire marcher l'extension, reprenez les étapes une par une... il est souvent plus simple de tout reprendre à zéro !

Si tout marche, vous pouvez passer à l'étape suivante, qui est la préparation de l'internationalisation, aussi appelée *i18n*.

B.A.-BA URL `chrome://`, `file://`...

Mozilla utilise plusieurs protocoles pour accéder aux fichiers. Tandis que certains sont généralistes et se retrouvent dans les autres navigateurs (`file://` pour les fichiers locaux, `http://`, `ftp://`... pour ne citer que les plus connus), il en existe des particuliers, notamment l'URL `chrome://`. Elle sert à accéder aux extensions enregistrées dans le fichier `chrome.rdf` et à tous leurs fichiers.

Globalement, vous utiliserez `chrome://nomdupaquet/content` ou `locale` ou `skin/fichier.xul`. Ainsi, une fois que vous distribuerez l'application, vous saurez que tous les fichiers seront mis dans le dossier *chrome* du navigateur et vous n'aurez pas à vous préoccuper de l'endroit où est installé le navigateur. De plus, les URL `chrome` ne dépendent pas de la langue, comme nous allons le voir au paragraphe suivant.

POUR ALLER PLUS LOIN Vos propres protocoles

Si c'est une relecture du livre que vous faites (et seulement dans ce cas, car le lien proposé est très technique), vous pouvez essayer de créer vos propres protocoles ! En modifiant un peu l'exemple, il serait possible d'utiliser une URL de type :

`xchrome://xulforum/content/`
pour accéder à l'extension.

► http://kb.mozillazine.org/Dev:_Extending_the_Chrome_Protocol

Intégration d'une DTD et début de l'internationalisation

On pourrait se demander l'utilité d'aborder si tôt l'internationalisation, alors que les traductions viennent en général après la programmation. En fait, si l'on ne prend pas en compte l'internationalisation dès le départ, il faut un travail énorme de modifications et de retouches sur les fichiers XUL pour l'intégrer. L'internationalisation n'est pas le fait de traduire notre application : c'est le fait de rendre les traductions futures possibles et aisées.

CULTURE Une DTD, concrètement, qu'est-ce ?

Nous n'utilisons en fait ici qu'une seule fonctionnalité des DTD : les entités. Il y en a bien plus. Bien que le système des DTD soit controversé car trop imprécis quant à sa définition d'un document XML, nous pourrions néanmoins voir un exemple de DTD, pour le petit fichier XML illustré par la copie d'écran donnée en bas de page :

```
<!ELEMENT livres (livre)> ①
<!ELEMENT livre (#PCDATA)> ②
<!ATTLIST livre titre CDATA #REQUIRED> ③
<!ATTLIST livre auteur CDATA #IMPLIED> ④
```

Des explications s'imposent.

Pour la première balise, `<livres>`, ① on précise que ses enfants doivent être des balises `livre`. On aurait pu écrire `(livre+)` pour préciser qu'il doit y en avoir au moins un. Les autres modificateurs ? (0 ou 1 occurrence), * (0 ou plus) sont aussi possibles.

Pour la balise `<livre>` ②, on précise qu'elle contient du texte (*parsed character data*) qui doit être analysé (il pourrait y avoir d'autres balises dedans). On aurait aussi pu mettre comme contenu ANY (pour tout type de contenu) ou EMPTY pour une balise vide (comme `
`).

La troisième ligne ③ précise qu'un attribut `titre` est requis sur l'élément `<livre>` et que cet attribut contient du texte non analysé. La dernière ligne ④ est

similaire à la précédente, à ceci près toutefois que l'attribut `auteur` n'est pas indispensable.

Une fois prête, une DTD s'insère dans un document XML selon la syntaxe suivante :

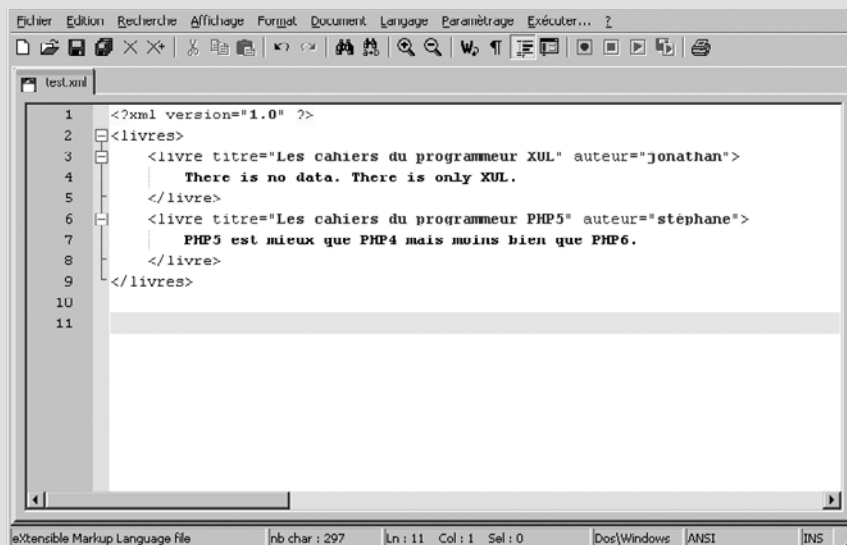
```
<!DOCTYPE nomdutagprincipal SYSTEM
"http://www.example.com/ma.dtd">
```

Ceci est dans le cas d'une DTD système. Pour une DTD publique, on écrira (dans le cas XHTML 1.0 strict) :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-strict.dtd">
```

Cette fois-ci, la DTD est publique, donc son adresse est précédée d'un identifiant, public lui aussi, permettant au navigateur de comprendre directement de quel type de document il s'agit. Au cas où ce dernier ne « comprendrait pas », il aurait toujours comme solution de secours d'aller chercher la bonne DTD à l'adresse indiquée en suivant l'URL.

Nous utiliserons surtout la première forme pour les documents XUL. Il y a un tutoriel assez succinct disponible sur le site *w3schools* qui vous permettra néanmoins d'assimiler les concepts de base de DTD, trop rapidement évoqués ici.



L'internationalisation se fait au travers d'entités XML définies dans une DTD externe, comme les entités HTML ´ pour le caractère « é », pour une espace insécable (nbsp évoquant les termes anglais *non-breaking space*), etc. Dans notre cas, au lieu d'écrire « Identification », nous écrirons &ident.titre; et nous définirons pour chaque langue l'entité ident.titre. Le navigateur se chargera ensuite de retrouver la bonne langue, de localiser le fichier contenant les entités et d'associer la bonne valeur à chaque entité.

Dossier locale

Comme nous avons créé un dossier content, nous allons créer un dossier locale. Ensuite, pour chaque langue, nous allons créer un sous-dossier portant le nom de la langue. Ainsi, depuis la racine de l'extension, on trouvera les dossiers locale/fr-FR, locale/en-US par exemple. Dans chaque sous-dossier devra se trouver un fichier contents.rdf. Ce fichier, que nous allons placer dans le dossier fr-FR, sera, à peu de choses près le même que celui créé précédemment.

```
<?xml version="1.0" ?>

<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <RDF:Seq about="urn:mozilla:locale:root">
    <RDF:li resource="urn:mozilla:locale:fr-FR" />
  </RDF:Seq>

  <RDF:Description about="urn:mozilla:locale:fr-FR"
    chrome:displayName="Pack français pour XUL Forum"
    chrome:author="Jonathan Protzenko"
    chrome:name="fr-FR">
    <chrome:packages>
      <RDF:Seq about="urn:mozilla:locale:fr-FR:packages">
        <RDF:li resource="urn:mozilla:locale:fr-FR:xulforum" />
      </RDF:Seq>
    </chrome:packages>
  </RDF:Description>

</RDF:RDF>
```

On remarque d'abord quelques différences minimales : utilisation de urn:mozilla:locale dans les attributs et de fr-FR comme nom de locale. Une précaution sera de prévoir deux locales : une en-US qui sera présente par défaut dans tous les navigateurs et une autre fr-FR pour les utilisateurs francophones.

La locale choisie sera celle qui aura été placée le plus haut dans la liste de préférences de l'utilisateur.

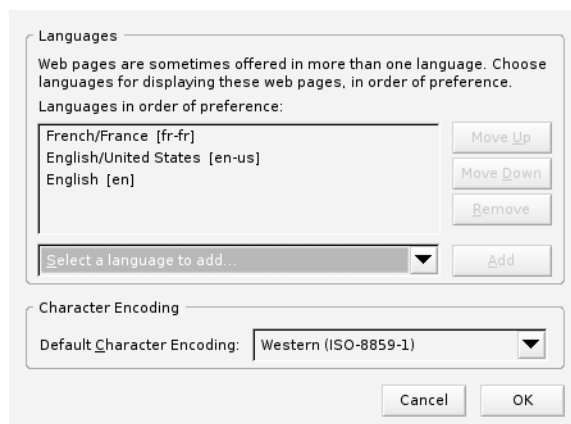
OUTILS Édition de locales

Pour un maximum d'efficacité, pensez à prévoir deux fenêtres lorsque vous éditez votre fichier XUL. Une pour la DTD et l'autre pour le fichier XUL. Ainsi, vous pourrez gérer l'internationalisation au fur et à mesure, ce qui vous évitera de devoir revenir péniblement sur votre code après. Ne commettez par l'erreur de vous dire que vous y reviendrez plus tard... c'est encore plus pénible !

B.A.-BA Locale

Une *locale* est un attribut qui permet de définir la langue utilisée. Elle se compose généralement de deux parties : la première est la langue, la deuxième concerne la localisation. Une première partie sera par exemple en (anglais), fr (français) ou zh (chinois) ; une deuxième partie sera FR (France), CA (Canada) ou GB (Grande-Bretagne), US (États-Unis), ou encore CN (République populaire de Chine), TW (Taiwan). Là où pour le français ou l'anglais les différences sont minimales (en en-US on écrit « save to disk » et en-GB « save to disc »), pour du chinois par exemple, la signification est totalement différente : les encodages et les idéogrammes ne sont pas les mêmes selon que l'on se trouve en République Populaire de Chine ou à Taiwan ! Les locales sont utilisées par exemple dans un environnement Linux pour définir la langue à choisir dans les applications.

Figure 4-2
Choix des locales
préférées dans Firefox



Ensuite, la section `<chrome:packages>` indique, via le classique `<RDF:Seq...>` `<RDF:li>`, quel paquet est concerné par ces fichiers de langue. Ici on précise que c'est notre paquet XUL Forum, déjà présent dans le navigateur.

Ces fichiers `contents.rdf` sont très figés et ne varient généralement que très peu d'une extension à l'autre. Le plus dur est de les recopier sans se tromper !

Modification du fichier XUL et ajout d'une DTD

Il nous faut maintenant éditer simultanément les fichiers `locale/fr-FR/xulforum.dtd` et `content/xulforum.xul`. Nous garderons une seule DTD, soit un seul fichier pour toutes les pages de XUL Forum. Ceci évitera bon nombre de problèmes par la suite, lorsque nous aurons à jongler entre différents fichiers.

```
<!ENTITY ident.titre "Identification">
<!ENTITY ident.xf-ident_nom "Votre nom">
<!ENTITY ident.xf-ident_pass "Votre mot de passe">
<!ENTITY fenetre.titre "XUL Forum !">
```

Pour ceux qui ont déjà travaillé sur des DTD, la structure est familière. Pour ceux qui en sont moins coutumiers, on définit les entités en utilisant la structure `<!ENTITY nomdel'entité "Valeur de l'entité">`. Le format `nomglobal.nomparticulier` est entièrement facultatif, mais il permet de mieux s'y retrouver et d'éviter des confusions. Ici, le cadre est le cadre d'identification. Il a un titre et les deux champs ont leurs attributs `label` spécifiés.

Nous allons maintenant apporter deux types de modifications au fichier XUL : d'abord ajouter un lien vers la DTD, puis remplacer les textes par les entités qui leur correspondent.

Voici le fichier XUL modifié :

xulforum.xul, avec prise en charge de l'internationalisation

```
<?xml version="1.0" ?>
<?xml-stylesheet href="chrome://global/skin" type="text/css" ?>

<!DOCTYPE window SYSTEM "chrome://xulforum/locale/xulforum.dtd">

<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul" title="&fenetre.titre;">
...
<groupbox>
  <caption label="&ident.titre;" />
...
  <description value="&ident.xf-ident_nom; : " />
  <textbox id="xf-ident_nom" />
...
  <description value="&ident.xf-ident_pass; : " />
  <textbox id="xf-ident_pass" type="password" />
...
</window>
```

Une nouvelle ligne a tout d'abord fait son apparition après le prologue XML. Elle sert à référencer la DTD, permettant de valider ce document. Ici, utiliser une DTD pour décrire la structure du document n'aurait pas de sens, puisque XUL est déjà défini. Ce n'est pas comme si nous définissions notre propre format XML. La DTD sert juste à définir nos propres entités. Si vous avez l'habitude de valider vos documents HTML, cette ligne vous sera familière. Le `<!DOCTYPE HTML` est devenu `<!DOCTYPE window` : la DTD s'applique au *tag* `window` car elle est déclarée comme `<!DOCTYPE nomdutagprincipal`.

CULTURE Attributs du tag `<window>`

Nous avons ajouté un attribut `title`, qui signifie « titre », à notre fenêtre. Il en existe d'autres, tout aussi utiles, comme `persist`. Il prend une liste de valeurs, séparées par des espaces et permet d'indiquer les attributs de la fenêtre dont il faudra se souvenir pour la prochaine ouverture : `screenX` (coordonnées X de la position de la fenêtre), `screenY`, `width` (largeur), `height` (hauteur)...

Ces attributs peuvent aussi être utilisés directement sur le tag `window`. Voici un exemple qui mélange les deux utilisations :

```
<window width="800" height="600"
  persist="screenX screenY" ...>
```

L'attribut `persist` peut être utilisé sur n'importe quel élément XUL.

Ce qui est remarquable ici, c'est que l'on n'a pas besoin de déterminer le langage choisi par l'utilisateur. Mozilla se charge de compléter le chemin vers la DTD avec le bon dossier et transforme `chrome://xulforum/locale/madtd.dtd` en `chrome://xulforum/locale/fr-FR` ou `en-US` ou `autre_chose/madtd.dtd`. Enfin, on remplace tous les textes par leurs entités correspondantes. Attention aux fautes de frappe !

ATTENTION DTD sur des fichiers séparés

Une idée intéressante serait de séparer les entités sur différents fichiers : un fichier `xulforum.dtd` pour les entités correspondant à `xulforum.xul`, un autre pour une autre page, etc.

Un problème se pose alors lorsqu'on utilise des éléments globaux dans différentes pages. Ainsi, les entités du menu devront être utilisées dans le premier écran d'identification, puis dans la fenêtre principale du forum, éventuellement dans des boîtes de dialogue...

Il vaut donc mieux garder toutes les entités dans un fichier unique et les séparer en blocs dans le fichier DTD, pour en améliorer la lisibilité.

Chrome.rdf

Maintenant, de même que nous avons enregistré l'extension auprès du navigateur, nous allons ajouter une nouvelle ligne au fichier `installed-chrome.txt`.

```
| locale,install,url,file:///mnt/data/Livre/xulforum/locale/fr-FR/
```

Là encore, vous pouvez modifier cette ligne si vous développez votre extension dans le dossier `chrome` du navigateur. Supprimez le fichier `chrome.rdf` et redémarrez Firefox.

Il est rare que tout fonctionne du premier coup ! Souvent, une faute de frappe, une erreur de réglage font que l'extension ne gère pas la DTD demandée. Tout d'abord, vérifiez que vous avez bien la locale `fr-FR` en tête de liste dans vos préférences de langue. Si ce n'est pas le cas, le navigateur peut chercher une locale `en-US` et ne pas la trouver. Ensuite, vérifiez l'encodage de la DTD. Un accent qui ne correspond pas au bon encodage peut être fatal et provoquer une erreur. Enfin, vérifiez le chemin de la DTD dans votre fichier `xulforum.xul`.

Si tout fonctionne, vous pouvez respirer : le plus gros est fait ! La structure de base de l'application est posée et, dans le prochain chapitre, nous ne parlerons que de XUL.

POUR ALLER PLUS LOIN Accessibilité

En toute rigueur, il faut également définir les touches de raccourci pour les éléments XUL, en fonction de la langue. Pour un éventuel bouton *Valider*, on écrira ainsi dans la DTD :

```
<!ENTITY bouton.valider "Valider">
<!ENTITY bouton.valider.accesskey "V">
```

Puis dans le fichier XUL :

```
<button accesskey="&bouton.valider.accesskey;"
label="&bouton.valider;" />
```

Dans la majeure partie des cas, la lettre V sera soulignée et une pression sur cette même touche amènera le *focus* sur ce bouton.

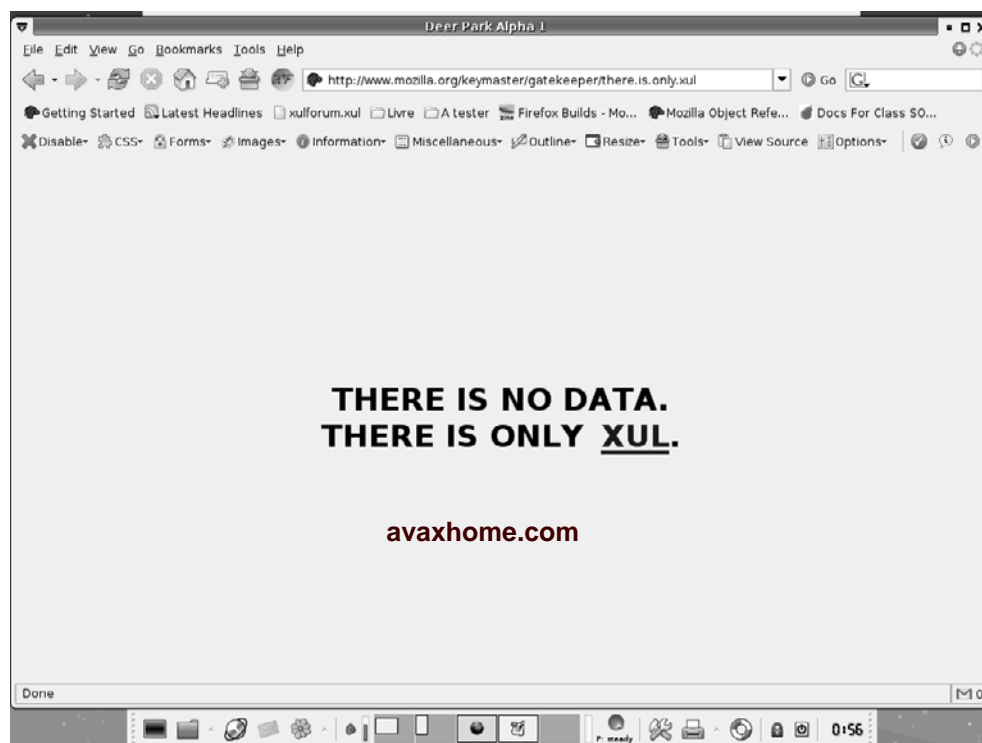
En résumé...

Nous avons, dans ce long chapitre, bien avancé la mise en place de l'application :

- les différents fichiers ont été intégrés au cœur même de Mozilla pour en faire une extension interne accessible via l'URL `chrome://` ;
- un système d'internationalisation a été mise en place dès le départ, avec là encore une intégration au sein même du navigateur.

Le seul regret que nous pouvons avoir est de ne pas encore assez parler de XUL ! Dans ce chapitre, ce sont surtout les technologies connexes qui se sont imposées. Dans le chapitre qui va venir, nous allons construire la fenêtre principale de XUL Forum et un peu mettre de côté les éléments qui concernent l'intégration dans Mozilla.

chapitre 5



XUL avancé : la fenêtre principale

L'écran d'authentification n'était qu'une introduction. Dès maintenant, le gros du travail peut commencer !

SOMMAIRE

- ▶ Structure globale : séparation des grandes parties de l'interface
- ▶ Plus de clarté : répartition du code sur plusieurs fichiers
- ▶ Création de l'interface en détail avec les différents éléments

MOTS-CLÉS

- ▶ Overlays
- ▶ Splitters
- ▶ Arbre hiérarchique

CULTURE LXR ?

LXR signifie « Linux Cross Reference ». C'est un outil développé initialement pour consulter les sources du noyau Linux via une interface web. Il permet de mettre en relation les différentes pages contenant du code source C, en permettant par exemple d'aller à la définition d'une fonction, d'aller directement à l'endroit où est déclarée une structure, etc.

LXR est utilisé par le projet Mozilla pour naviguer entre les différentes branches (branche de développement, branche stable via une interface web) et pour consulter facilement du code éclaté sur plusieurs parties de l'ensemble Mozilla.

► <http://lxr.mozilla.org>

Maintenant que la syntaxe de XUL vous est plus familière, nous pouvons attaquer le codage de l'écran principal. Déjà introduit dans le chapitre 2, il peut se découper en grands ensembles. Nous allons partir du plus généraliste pour nous orienter vers le plus détaillé : d'abord un découpage de l'interface, puis un travail de précision pour bien placer les différents éléments.

La structure globale

Découpage avec les principales boîtes

Sur le schéma ci-contre, les grands éléments de l'écran principal de XUL Forum sont mis en relief. En fait, l'écran principal ressemble fortement à la fenêtre principale de Mozilla. La structure est éprouvée et permet un maximum d'efficacité : d'abord, en haut, les menus ; ensuite, la barre d'outils, contenant les différents boutons, pour la plupart des raccourcis vers des éléments de menu ; au milieu, les deux grandes parties, séparées par une poignée de redimensionnement ; enfin, en bas, la barre de statut.

Dans l'absolu, l'empilement des différents éléments implique l'utilisation d'une boîte verticale. Dans la pratique, c'est la fenêtre principale, orientée verticalement, qui jouera ce rôle : il est inutile de rajouter une boîte supplémentaire. Les deux boîtes verticales « membres » et « forum » seront séparées par une poignée pour permettre à l'utilisateur de redimensionner l'espace ; ces deux boîtes verticales seront elles-mêmes placées dans une boîte horizontale.

ASTUCE Fichier XUL principal de Mozilla, sur LXR

Si vous voulez regarder comment se structure l'interface principale du navigateur, vous pouvez aller consulter le fichier désigné à l'URL suivante :

► <http://lxr.mozilla.org/mozilla/source/browser/base/content/browser.xul>

Bien qu'il soit destiné à être traité par le préprocesseur C, avec notamment l'utilisation de `#include` en fonction des différentes plates-formes, il reste néanmoins lisible. Son contenu s'éclaircira au fur et à mesure du livre. Pour l'instant, la structure d'ensemble devrait être compréhensible, ainsi que les attributs des éléments que nous allons voir.

En fait, il faut comprendre que la fenêtre principale du navigateur est un fichier XUL tout à fait standard, certes un peu complexe, mais accessible comme tout document XUL via une URL chrome. Lancer Firefox revient à le faire pointer vers l'URL `chrome://browser/content`. Si vous l'avez déjà lancé, vous pouvez aller à cette adresse via la barre d'adresses et vous verrez le navigateur dans le navigateur. Pour la suite Mozilla, le fichier XUL du navigateur est `chrome://navigator/content`.

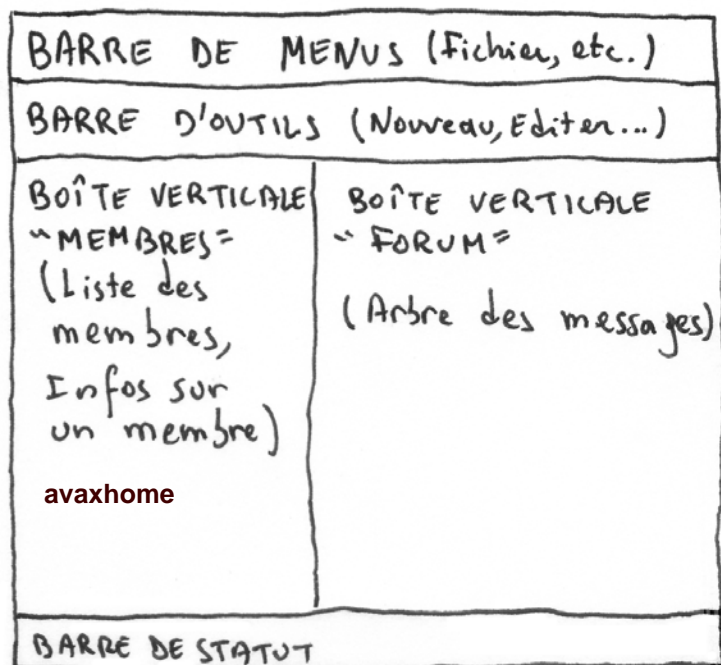


Figure 5-1
Les ensembles de l'écran principal mis en relief

Voici donc un aperçu global du document :

```
<?xml version="1.0" ?>
<?xml-stylesheet href="chrome://global/skin" type="text/css" ?>

<!DOCTYPE window SYSTEM
  "chrome://xulforum/locale/xulforum.dtd">

<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul"
  title="&index.fenetre_titre;">

  <!-- ici la barre de menu -->
  <!-- ici la barre d'outils -->
  <hbox>
    <!-- ici la partie de gauche pour la liste des membres -->
    <!-- élément séparateur -->
    <!-- ici la partie de droite pour l'arbre des messages -->
  </hbox>
  <!-- espaceur -->
  <!-- ici la barre d'état -->

</window>
```

La structure n'a finalement que peu changé par rapport au fichier précédent. Celui-ci s'appelle maintenant `index.xul` et intègre toujours la DTD globale de l'application. Il est toujours placé dans le dossier `content` de l'extension et nécessitera un redémarrage de votre navigateur pour être disponible.

RAPPEL Commentaires en XUL

Comme en XHTML, les commentaires en XUL sont encadrés par des balises `<!--` et `-->`. N'hésitez pas à abuser des commentaires, surtout lorsque le niveau d'imbrication des balises commence à être important. Vous serez alors heureux, lorsque vous reviendrez sur votre code, de pouvoir en comprendre rapidement la structure, la raison pour laquelle vous aviez placé tel ou tel élément, ou encore les différentes propriétés d'un attribut bizarre. Pensez aussi à la personne qui vous lira, le futur mainteneur de votre code, pour qui il est toujours heureux de laisser quelques commentaires !

Séparation en overlays

La fenêtre principale se compose d'un nombre assez important d'éléments. Il serait bien sûr possible d'imbriquer tous les éléments dans le même fichier, mais le niveau de profondeur deviendrait vite effrayant et un fichier de cette taille serait difficile à gérer. Une technique plus élégante serait d'éclater le contenu de la fenêtre principale sur plusieurs fichiers : l'un contiendrait tous les menus, la barre d'outils et la barre d'état, un autre l'affichage des éléments du forum et un dernier ce qui concerne les membres.

Comment répondre simplement à ce problème ? En utilisant la solution XUL des overlays. Ces derniers s'intègrent de manière transparente dans le document XUL, en effectuant une sorte « d'ajout » au document original.

Dans le document principal

Voici le document réduit au minimum que nous allons garder pour `index.xul`.

`index.xul` n'est plus qu'un squelette, amené à être rempli par les overlays

```
<?xml version="1.0" ?>
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<?xul-overlay href="chrome://xulforum/content/
  ➤ index-barres-overlay.xul"?>

<?xul-overlay href="chrome://xulforum/content/
  ➤ index-membres-overlay.xul"?>

<?xul-overlay href="chrome://xulforum/content/
  ➤ index-forum-overlay.xul"?>
```

On inclut chaque *overlay* grâce à une directive XML.

```
<!DOCTYPE window SYSTEM "chrome://xulforum/locale/index.dtd">
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul"
  title="&index.fenetre_titre;">
  <toolbox id="xf-index-toolbox" />

  <hbox>

    <vbox id="xf-index-vbox-membres" />

    <splitter collapse="before" ><grippy /></splitter>

    <vbox id="xf-index-vbox-forum" />

  </hbox>

  <statusbar id="xf-index-statusbar" />

</window>
```

- ◀ La barre d'outils devra contenir les menus et les boutons. Son ID est indispensable au bon fonctionnement de l'*overlay*.
- ◀ Les éléments doivent être placés dans une boîte horizontale.
- ◀ Cette boîte verticale contiendra les deux onglets, un pour la liste des membres, l'autre pour les informations relatives au membre courant. Elle sera dans la partie gauche de l'écran.
- ◀ On ajoute le séparateur avec, au milieu, la poignée (affichée sous Mozilla uniquement).
- ◀ Cette boîte contiendra l'arbre avec les messages du forum, dans la partie droite de l'écran.
- ◀ Une barre de statut, indispensable à toute application professionnelle, contiendra les messages d'information et la barre de progression, qui seront ses éléments fils.

ASTUCE Tags complets et forme abrégée

En XML, la façon de présenter vos *tags* dépend de ce qu'ils contiennent. De manière générale, si `<tag1 attribut1="xxx"></tag1>` ne contient aucun élément fils, il peut s'abrégé en `<tag1 attribut1="xxx" />`. Une fois le document XML analysé, la signification en est la même pour l'analyseur (le *parser*). Ainsi, dans l'exemple, des éléments qui ne se retrouvent généralement qu'accompagnés d'éléments fils sont abrégés, puisque leur contenu sera défini dans les *overlays*.

ATTENTION Nommage des fichiers, ID

Pour les attributs des éléments, nous avons choisi de les faire commencer par `xf` (comme XUL Forum), puis de les faire suivre du nom de la fenêtre et, enfin, de leur attribuer un nom parlant. Pour les fichiers, le problème est le même : il faut trouver des noms qui évitent la confusion tout en restant facilement identifiables. Ici, nous utiliserons : `nomdufichier.xul`, puis `nomdufichier-overlay-titreoverlay.xul` pour les *overlays*. Libre à vous de choisir une autre convention de nommage ; vous devrez simplement penser à modifier les références aux fichiers dans les exemples de code.

CULTURE Les attributs du tag splitter

L'élément `splitter` est rendu différemment selon que l'on utilise Mozilla ou Firefox. Sous Firefox, il est généralement constitué de deux barres continues ; sous Mozilla, on retrouve au centre la poignée (le *grippy*). Si l'on clique sur cette poignée ou que l'on double clique sur le séparateur avec Firefox, il se rabat et c'est la partie droite qui prend tout l'espace... ou la partie gauche !

L'attribut `collapse` (indispensable) indique justement quel comportement adopter : soit les éléments avant doivent être cachés (c'est notre cas), soit ce sont les éléments après (valeur *after*) qui doivent l'être.

ATTENTION Firefox/Mozilla

Il existe des différences notables entre Firefox et Mozilla quant à l'utilisation de XUL. L'une des principales différences se situe dans la gestion des barres d'outils : Firefox utilise l'élément `<toolbarpalette>` pour rendre les barres d'outils personnalisables, tandis que Mozilla n'utilise que `<toolbar>`. Firefox n'a pas non plus la possibilité de réduire les barres d'outils avec les poignées situées à gauche. De même, les séparateurs `<grippy>` ne sont pas « rendus » avec Firefox. Lorsque la différence sera trop importante pour ne pas être prise en compte, elle sera signalée sous forme de remarque. Pour l'instant, les problèmes de compatibilité se détectent facilement en testant le fichier dans la suite Mozilla et le navigateur Firefox.

Dans les fichiers overlay

Aucun de ces éléments n'est rempli. Nous les avons simplement mis pour indiquer leur emplacement dans le document racine, mais leur contenu sera défini dans les fichiers overlays.

Les trois lignes `<?xml-overlay>` placées en début du fichier `index.xul` indiquent à l'interpréteur qu'il doit intégrer le contenu des overlays spécifiés dans le document principal. Ainsi, les éléments portant les mêmes attributs ID dans l'overlay et dans le fichier `index.xul` seront fusionnés pour donner l'index de XUL Forum. Nous retrouverons donc dans le fichier `index-barres-overlay.xul` quelque chose ressemblant à :

```
<?xml version="1.0" ?>
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul">
  <toolbar id="index-toolbox">
    <!-- les éléments pour le menu, etc. -->
  </toolbar>
  <statusbar id="index-statusbar">
    <!-- ici quelque chose comme du texte, une barre de
      progression, etc. -->
  </statusbar>
</overlay>
```

Un overlay se présente comme un fichier classique. Seule la balise racine `<window>` devient `<overlay>`. Lorsque le navigateur « inclut » les overlays au document principal, il analyse les attributs `id` des éléments appartenant à l'overlay principal. Le moteur fusionne ensuite les deux fichiers : l'overlay et le fichier appelant l'overlay. Il applique pour ce faire deux règles à ne pas oublier :

- si un élément de l'overlay possède le même `id` qu'un élément du fichier principal, les sous-éléments du premier viennent s'ajouter aux sous-éléments du second ;
- les sous-éléments du premier seront placés à la suite des sous-éléments du second.

La règle s'applique sur tous les éléments de l'overlay dont l'attribut `id` est le même qu'un élément du fichier appelant. Les règles sont valables à tous les niveaux : les sous-éléments d'un sous-élément, en supposant qu'ils ont les mêmes `id`, seront également placés à la suite.

On peut illustrer ce principe par la création des menus, pour lesquels les overlays sont abondamment utilisés. Le fichier principal peut spécifier les éléments de menu et les sous-éléments qui lui sont propres et, à condition d'avoir les bons attributs `ID`, l'overlay peut ajouter après des menus ses propres menus et, à la suite des sous-menus, encore ses propres sous-menus.

OUTILS Aller-retours entre l'éditeur et le navigateur

Maintenant que notre extension est dans le dossier `chrome://`, il faut redémarrer le navigateur pour qu'il prenne en compte les modifications. À la longue, ceci peut devenir extrêmement pénible ; c'est

pourquoi il existe quelques astuces pour faciliter la vie du développeur. La première consiste à activer quelques « fonctionnalités cachées » :

<code>user_pref("nglayout.debug.disable_xul_cache", true);</code>	◀ Désactive le cache XUL.
<code>user_pref("nglayout.debug.disable_xul_fastload", true);</code>	◀ Lié au précédent, facultatif.
<code>user_pref("javascript.options.strict", true);</code>	◀ Active l'affichage de toutes les erreurs d'exécution.
<code>user_pref("javascript.options.showInConsole", true);</code>	◀ Active l'affichage de toutes les erreurs de syntaxe.
<code>user_pref("browser.dom.window.dump.enabled", true);</code>	◀ Active le dump sur la console, servira pour le chapitre sur JavaScript... indispensable !

Ces lignes sont à ajouter dans votre fichier `prefs.js`, situé dans `~/.mozilla/firefox/default.XXX/` sous Unix avec Firefox et dans `Application Data/Mozilla/Profiles/dossierduprofil` depuis le dossier de l'utilisateur (typiquement : `c:\documents and settings\jonathan`) sous Microsoft Windows avec la suite Mozilla.

Elles permettent d'activer des fonctionnalités utiles de *debug*, comme indiqué en commentaires. Une fois les préférences ajoutées, vous pouvez utiliser une ligne de JavaScript comme ceci pour ouvrir l'extension dans une autre fenêtre :

```
open("chrome://xulforum/content", null,
"toolbars: no, statusbar : no")
```

Vous pouvez inclure cette ligne de code dans une balise `<script>` dans une page HTML, ou tout simplement la taper dans la console JavaScript (« outils »,

« console JavaScript » dans les menus). Vous obtenez ainsi deux fenêtres : une avec le navigateur pour visualiser de la documentation, du code source (pensez à utiliser l'extension *webdeveloper* pour ouvrir vos sources dans un nouvel onglet !) et une autre avec l'extension. Comme le cache XUL est désactivé, vous n'avez qu'à recharger la page correspondant à XUL Forum pour voir les modifications apparaître, en pressant la touche F5.

Cette méthode a cependant des inconvénients, le plus important étant l'utilisation de techniques de développement dans le même profil que celui destiné au Web. Vous pouvez, si vous le désirez, lire l'article du wiki du site XulFr sur ce sujet, qui propose l'utilisation de deux Firefox en même temps, avec modification des binaires, ou l'utilisation de deux profils séparés.

► <http://xulfr.org/wiki/ConfigurerMozillaPourDevelopper>

Pour l'instant, ce sont nos propres documents qui représentent le fichier appelant et le fichier overlay. Quand nous intégrerons l'application « dans » le navigateur, nous définirons de nouveaux overlays, mais cette fois-ci, le fichier appelant sera le navigateur et nous y ajouterons nos propres sous-menus par exemple.

Le principe des overlays s'incarne en deux grands types distincts d'utilisation. Tout d'abord, il permet de séparer des pages complexes en plusieurs fichiers, comme nous l'avons fait précédemment. La lisibilité et la clarté du code s'en trouvent grandement améliorées. Ensuite, comme nous l'avons esquissé précédemment, les overlays permettent de regrouper des éléments communs à plusieurs fichiers dans un overlay, puis, de ne garder que les éléments spécifiques à chaque page. Ceci peut être utile pour un menu ou une barre de statut qui, à peu de choses près, seront les mêmes pour toutes les pages.

ATTENTION Overlays distants

Pour des raisons de sécurité évidentes (par exemple, l'accès à des fichiers locaux serait possible pour un site internet !), il n'est pas permis de charger un overlay depuis une source tierce. Autrement dit, tous vos overlays doivent être sur le même domaine que le fichier XUL principal. Un simple fichier XUL local incluant un overlay commençant par une URL de type

`http://localhost/` ne marchera que si vous accédez au fichier XUL principal via l'URL `localhost`, mais ne fonctionnera pas si vous utilisez l'adresse numérique équivalente `127.0.0.1` ! Vous pouvez visualiser l'historique de ce bug sur le site de Mozilla, BugZilla, qui les référence à l'adresse :

► https://bugzilla.mozilla.org/show_bug.cgi?id=159450.

```
<!-- fichier exemple.xul -->
<?xml version='1.0' ?>
<?xul-overlay href="http://127.0.0.1/~jonathan/overlay.xul" ?>
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  < vbox id="exemple">
  </vbox>
</window>

<!-- fichier overlay.xul -->
<?xml version='1.0' ?>
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  < vbox id="exemple">
    <description>Overlay ici !</description>
  </vbox>
</overlay>
```

L'overlay fonctionne si l'on visualise `http://127.0.0.1/~jonathan/exemple.xul` mais échoue si l'on utilise `http://localhost/~jonathan/exemple.xul`.

Ceci est donné à titre informatif si par exemple vous décidez de créer une page XUL et de la placer sur un serveur distant, comme pour la page d'inscription de XUL Forum.

Un autre avantage pour les overlays est leur implication dans le mécanisme d'extensions Mozilla. Une extension doit, pour s'intégrer dans le navigateur, rajouter des éléments : barres d'outils, entrées dans les menus, etc. Ceci se fait au travers des overlays. On enregistre les fichiers contenant les overlays auprès du navigateur via le fichier `contents.rdf`, puis on peut étendre les éléments constituant l'interface si l'on connaît leurs attributs ID. Par exemple, l'ID de la barre de statut, en bas de la fenêtre du navigateur, est `status-bar`. Si vous connaissez l'extension « FoxyTunes » de Firefox, vous comprenez maintenant comment elle rajoute ses propres boutons dans la barre de statut !

ALTERNATIVE **Include()** et autres langages côté serveur

Comme nous l'avons vu en introduction, nous développons nos pages XUL de manière statique : elles ne sont pas créées dynamiquement. Il est également possible de les concevoir à l'aide d'un langage de pages web comme ASP ou PHP pour ne citer que les plus connus. Si votre application se destine uniquement à un contexte web, le mécanisme pourra être reproduit en utilisant des fonctions telles que `include()` en PHP par exemple. De manière plus générale, une fonction `afficherBarreDeMenu()` pourra être utilisée pour centraliser les éléments communs sur une page.

Dans le code source de Mozilla, c'est le préprocesseur C qui est appliqué aux fichiers XUL (et non au compilateur !). Il permet d'adapter les fichiers XUL à inclure, au final, en fonction de la plate-forme, ou même de séparer du code sans utilisation d'*overlays*. Le préprocesseur se charge de remplacer `#include fichierinclus.xul` par le contenu de `fichierinclus.xul`.

CULTURE **BugZilla**

BugZilla est le système de gestion de bogues (de *bugs* en anglais) développé d'abord par Netscape, puis repris par Mozilla. Au-delà d'une simple base de données, BugZilla permet de centraliser les rapports de bogues, les commentaires d'utilisateurs, les documents liés à un bogue et également de rassembler les demandes d'amélioration du logiciel. Par exemple, pour l'intégration du standard Xforms du W3C proposant une nouvelle façon de créer les formulaires, il existe un rapport de bogue à l'adresse :

► https://bugzilla.mozilla.org/show_bug.cgi?id=97806

qui centralise toutes les fonctionnalités nécessaires à la réalisation du support de XForms. Les commentaires sur ce rapport de bogue permettent en fait de suivre l'évolution de son implémentation.

► <http://bugzilla.mozilla.org>

Le fichier `index-barres-overlay.xul` : barres d'outils, de menu et de statut

Maintenant que l'interface est découpée en grands ensembles, nous pouvons attaquer la rédaction du premier fichier overlay : `index-barres-overlay.xul`. Il contiendra la barre de menu, la barre d'outils et la barre de statut. Nous pouvons d'ores et déjà énoncer sa structure.

```
<?xml version="1.0" ?>
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <toolbox id="xf-index-toolbox">
    <menubar id="xf-index-menubar">
      <!-- ici la barre de menu -->
    </menubar>
    <toolbar id="xf-index-toolbar">
      <!-- ici les boutons avec des images -->
    </toolbar>
  </toolbox>
```

```

<statusbar id="xf-index-statusbar">
  <!-- ici quelque chose comme du texte,
        une barre de progression... -->
</statusbar>
</overlay>

```

L'élément `<toolbox>` est une boîte verticale spéciale, qui place une poignée à gauche de la fenêtre pour chacun de ses éléments, dans le cas de la suite Mozilla. Dans le cas de Firefox ou autre, la poignée n'est pas disponible. Une `toolbox` contient, dans le cas du navigateur, les menus, la suite de boutons, la barre d'adresses, le *Bookmarks toolbar folder*, etc. Dans notre application, l'élément `toolbox` contiendra seulement un élément `menubar` et un élément `toolbar` pour la barre d'outils, qui contiendra des boutons comme : « nouveau », « éditer », etc.

Les menus

La première chose que l'utilisateur voit apparaître dans une application est la barre de menu. Que ce soit un éditeur de texte ou un navigateur, toute application digne de ce nom possède des menus. Pour ne pas être en reste, nous allons nous aussi ajouter nos menus à XUL Forum.

Pour ce faire, nous utiliserons quatre éléments : `<menubar>`, `<menu>`, `<menupopup>` et `<menuitem>`. Le premier est un conteneur horizontal : il est le premier à être placé dans la `<toolbox>`. Il contiendra les éléments `<menu>`. Le second représente le bouton en lui-même, comme le rectangle qui contient le texte *Fichier* dans un programme classique. Le `<menupopup>` est un conteneur qui s'affiche lorsque l'on clique sur un élément `<menu>`. Il contient les éléments du sous-menu représentés par des `<menuitem>`. Voici un exemple plus parlant :

► C'est le premier élément, la barre de menu. Tout ce qui concerne un menu sera placé dedans.

► Ceci est le premier élément de menu. Graphiquement, c'est le texte *Fichier* et son cadre autour.

► Ce *widget* structurel représente le *popup* : c'est en fait une fenêtre avec des éléments de menu. Il peut aussi être utilisé lors d'un clic droit.

► C'est un élément de menu fixe, qui ne contient pas d'autres sous-menus.

► Un séparateur, représenté graphiquement par une ligne horizontale.

```

<menubar id="xf-index-menubar">

  <menu label="Fichier">

    <menupopup>

      <menuitem label="Nouveau sujet" />
      <menuitem label="Editer le sujet" />

      <menuseparator />

      <menuitem label="Quitter" />
    </menupopup>
  </menu>

```

```

<menu label="Connexion">
  <menupopup>
    <menu label="Serveur">
      <menupopup>
        <menuitem label="Vérifier la connexion
          ➡ au serveur PHP" />
        <menuitem label="Vérifier la connexion
          ➡ au serveur LDAP" />
      </menupopup>
    </menu>
    <menuitem label="Revenir à l'écran
      ➡ d'authentification" />
  </menupopup>
</menu>
</menubar>

```

- ◀ Un deuxième élément de menu.
- ◀ Ici, il y a un autre niveau de profondeur : on remplace le *menuitem* par un menu.
- ◀ On reproduit le mécanisme.

Seule la barre de menu est mentionnée ici. Elle se trouve dans le fichier `index-barres-overlay.xul`. Pour la clarté de l'exemple, les chaînes de caractères sont incluses dans le fichier XUL, mais en réalité, elles sont placées dans une DTD à part, comme nous l'avons vu précédemment.

Ce menu n'est bien sûr qu'un exemple et n'est pas définitif : nous serons amenés à le modifier tout au long du développement. En fait, dans ce chapitre, nous construisons surtout une interface statique. Le menu sera remodelé, retravaillé en fonction des besoins, à l'aide de JavaScript et de CSS.

Chose surprenante, l'élément `<menubar>` est facultatif. En fait, vous pouvez tout à fait remplacer ce `<menubar>` par une `<hbox>`. Mais le rendu n'est alors pas du tout le même ! On trouve des boutons très fins, avec une flèche sur leur droite et un menu qui s'ouvre lorsque l'on clique dessus. C'est une utilisation qui peut servir, mais uniquement dans des cas très particuliers. À vous d'essayer !

POUR ALLER PLUS LOIN **RDF et les menus**

On devine aisément que l'écriture d'un menu devient longue et fastidieuse dès que l'on entre dans un niveau de profondeur supérieur à deux. L'indentation dans l'éditeur devient essentielle et, surtout, on se perd à la relecture du code dans des imbrications de menus, de `menupopups`, de `menuitems`, etc. Une recreation automatique du menu à partir d'un fichier décrivant sa structure serait une solution plus efficace : elle est possible au travers de la technologie RDF. On décrit le menu dans un fichier séparé et on crée le code XUL du menu à partir de ce fichier : ce sont les *templates*. En fait, nous avons vu dans ce chapitre la solution la plus « classique ». Lorsque vous aurez lu et compris le chapitre sur RDF, vous pourrez essayer cette solution alternative.

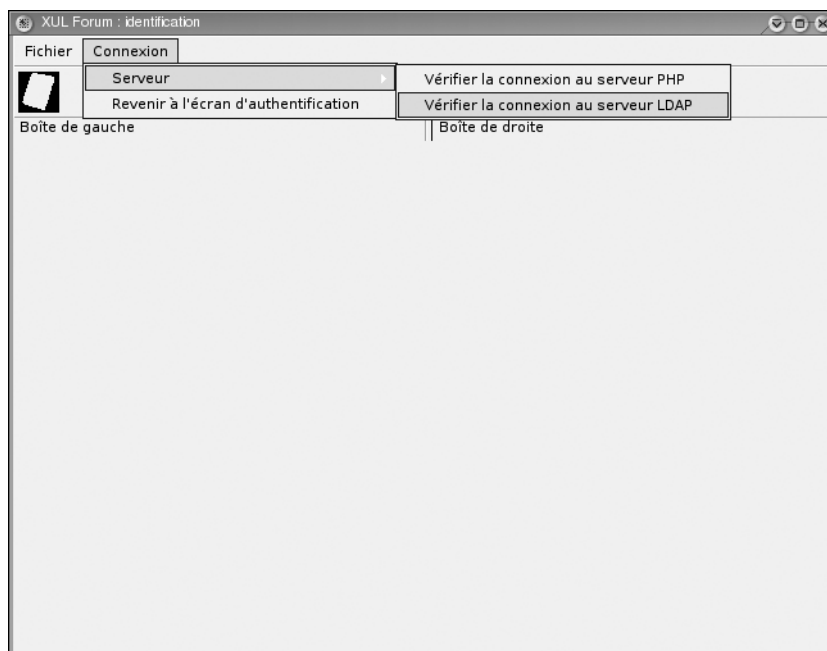


Figure 5-2
L'aspect de notre menu

Des menus plus évolués

Il existe d'autres fonctionnalités pour les éléments de menu : ils peuvent notamment arborer une case à cocher ou se comporter comme des boutons radio. Un menu de configuration sera un parfait exemple pour illustrer ceci : des options à choisir ou des cases à cocher sont tout à fait utilisables via un menu.

```
<menu label="Configuration">
  <menupopup>
    <menu label="Choix du type d'appel distant">
      <menupopup>
        <menuitem name="appel" type="radio" label="XML-RPC" />
        <menuitem name="appel" type="radio" label="SOAP" />
        <menuitem name="appel" type="radio" label="WSDL"
          checked="true" />
      </menupopup>
    </menu>
    <menuitem type="checkbox" checked="true" disabled="true"
      label="S'identifier d'abord auprès du serveur LDAP" />
  </menupopup>
</menu>
```

Nous introduisons de nouveaux attributs par rapport à l'exemple précédent. Détaillons-les ensemble :

- `disabled` permet, comme son nom l'indique, de désactiver un élément de menu (en fait, bon nombre d'éléments ont un attribut `disabled` comme les boutons ou les `textbox`) ;
- `type` permet de spécifier la nature de l'élément :
 - si la valeur est `radio`, alors un élément et un seul pourra être sélectionné au choix parmi les éléments portant le même attribut `name` ;
 - si la valeur est `checkbox`, alors l'élément de menu pourra être coché : une croix apparaîtra à gauche de l'élément ;
- `checked` permet, dans le cas d'un type `checkbox` de cocher directement l'élément, ou dans le cas d'un type `radio`, de choisir l'élément sélectionné par défaut.

Ceci permet une plus grande variété dans l'utilisation des menus. Si vous y tenez, vous pouvez même, en attendant le chapitre sur les CSS, utiliser cette structure pour placer une icône à gauche de l'élément `<menuitem>`.

```
<menuitem label="&index.menu_fichier_nouveau;"
  class="menuitem-iconic"
  image="chrome://xulforum/content/images/menu-new.png" />
```

B.A.-BA Mais où trouve-t-on tous ces attributs ?

Le développeur lisant cet ouvrage est en droit de se demander où trouver une référence exhaustive des composants XUL. Sachez que l'on recense plus d'une centaine de balises ! Or depuis le début du chapitre, les noms d'éléments se succèdent en rafale, sans forcément que l'on soit capable de les trouver seul.

Il faut savoir que la plupart ne sont utilisés qu'occasionnellement : les plus courants sont représentés dans le code de XUL Forum. La référence reste cependant le site xulplanet.com sur lequel vous trouverez une liste de tous les éléments XUL, XBL, XPCOM ou encore des propriétés CSS et consultable en ligne ! Vous avez aussi à votre disposition l'annexe des éléments graphiques les plus utilisés.

La méthode la plus courante est d'utiliser LXR pour examiner comment l'on procède pour obtenir tel ou tel comportement ou tel ou tel rendu. Le fichier principal a été présenté plus haut : il faut ensuite jongler entre les fichiers inclus, le JavaScript et les autres composants. Toutes les applications XUL sont également une mine d'informations : dès qu'une extension a un effet ou une organisation qui vous intéresse, n'hésitez pas à disséquer le fichier `.xpi` puis le `.jar`, pour voir comment l'auteur est arrivé au résultat qui vous préoccupe.

La page de référence de l'élément sur le site [xulplanet](http://xulplanet.com) permettra ensuite de bien comprendre l'utilité de chaque propriété.

Il existe enfin une page sur le Web qui montre les différentes utilisations des widgets XUL les plus courants ainsi que l'effet de leurs différents attributs. La combinaison des trois devrait vous permettre, une fois seul, d'arriver au résultat voulu ! Si vraiment vous ne trouvez pas, il reste le forum XUL de xulplanet.com, en anglais, ou celui de xulfr.org, la communauté francophone des utilisateurs de XUL.

L'astuce « ultime » consiste à utiliser une barre d'outils intégrée à Mozilla, pour aller en un clic à la référence d'un objet !

- ▶ <http://www.xulplanet.com/references/elemref/>
- ▶ <http://xulplanet.com/forum/>
- ▶ <http://lxr.mozilla.org/mozilla/source/browser/base/content/>
- ▶ <http://www.hevanet.com/acorbin/xul/top.xul>
- ▶ <http://xulfr.org>
- ▶ <http://xulfr.org/outils/>

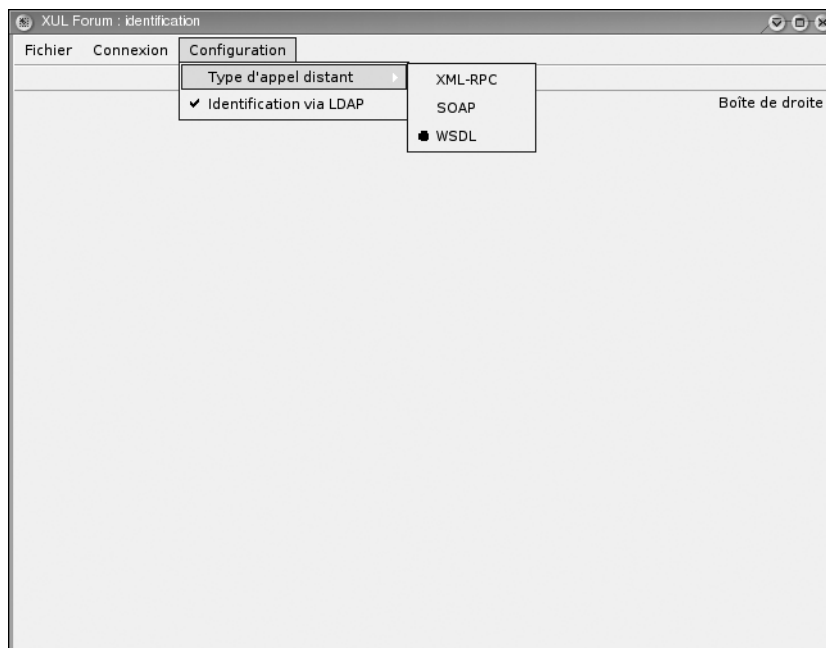


Figure 5-3
Le menu de configuration

Les menus sont de suite plus agréables à l'œil ! Mais cette méthode n'est pas recommandée ; nous en verrons une meilleure quand nous aborderons les CSS.

Nous reviendrons plus en détail sur les classes et les attributs de style propres à Mozilla, mais pour l'instant, ces deux attributs vous permettront de donner une « touche » plus agréable à l'application, qui est pour l'instant un peu austère. L'attribut `image` présuppose que vous avez créé un dossier `images` dans `content` et trouvé une icône qui convient.

Éléments communs à toutes les pages

Notre découpage en overlays est satisfaisant, mais n'est pas encore parfait. En effet, si l'on veut que la première page, celle permettant l'identification, puisse elle aussi utiliser les menus, on aura forcément les éléments « Nouveau sujet » et « Éditer le sujet » qui seront présents, alors qu'ils n'ont aucune utilité sur cette page. En fait, il faudrait séparer les éléments communs à toutes les pages de l'extension et les éléments spécifiques à chaque page. C'est ce que nous allons faire, en plaçant les menus dans un overlay particulier et en gardant les éléments spécifiques dans le fichier appelant.

Dans `index.xul`, on ne garde que les trois premiers éléments du menu fichier, spécifiques à cette page.

```
...
<?xul-overlay href="chrome://xulforum/content/menus-
overlay.xul"?>

<!DOCTYPE window SYSTEM "chrome://xulforum/locale/index.dtd">

<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
there.is.only.xul"
title="&index.fenetre_titre;">

  <toolbox id="xf-index-toolbox">
    <menubar id="xf-menubar">
      <menu id="xf-menu-fichier">
        <menupopup id="xf-menupopup-fichier">
          <menuitem label="&index.menu_fichier_nouveau;"
class="menuitem-iconic"
image="chrome://xulforum/content/images/
menu-new.png" />
          <menuitem label="&index.menu_fichier_editer;" />
          <menuseparator />
        </menupopup>
      </menu>
    </menubar>
  </toolbox>
...

```

Vous remarquerez que certains éléments ID ont fait leur apparition : ceci va s'expliquer par la nature du fichier `menus-overlay.xul` qui suit :

```
<?xml version="1.0" ?>
<!DOCTYPE overlay SYSTEM "chrome://xulforum/locale/index.dtd">
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/
there.is.only.xul">
  <toolbox id="xf-index-toolbox">
    <menubar id="xf-menubar">
      <menu id="xf-menu-fichier" label="&index.menu_fichier;">
        <menupopup id="xf-menupopup-fichier">
          <menuitem label="&index.menu_fichier_quitter;"
oncommand="window.close();"
id="xf-menu-fichier-quitter" />
        </menupopup>
      </menu>
    </menubar>
  </toolbox>
...

```

Les attributs ID spécifiés sont les mêmes que dans `index.xul` et on garde uniquement l'élément « quitter », qui se retrouvera sur toutes les pages. Comme les éléments de l'overlay sont ajoutés à la suite des éléments du fichier appelant, l'ordre des `<menuitem>` est conservé.

Ce découpage est plus « fin » que le précédent : le mécanisme d'overlays fusionne intelligemment fichier principal et fichiers overlays et la réutilisation de code est plus performante qu'avec le système précédent.

On peut ainsi faire passer dans le fichier `menus-overlay.xul` la plupart des options de menu de la première page, qui restent valides une fois arrivé sur `index.xul`. Par contre, on pourra considérer que l'option « S'identifier aussi auprès du serveur LDAP » concerne uniquement l'authentification : nous ne la garderons que dans `xulforum.xul` ; elle permettra de choisir si, au moment de l'identification, il faut vérifier que les informations données sont correctes auprès du serveur LDAP ou auprès du serveur PHP.

Le contenu du fichier `xulforum.xul`

```
<?xul-overlay href="chrome://xulforum/content/menus-overlay.xul"?>
<!DOCTYPE window SYSTEM "chrome://xulforum/locale/xulforum.dtd">
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul"
  title="&ident.fenetre_titre;" width="640" height="480"
  persist="width height">
  <toolbox id="xf-index-toolbox">
    <menubar id="xf-menubar">
      <menu id="xf-menu-configuration">
        <menupopup id="xf-menupopup-configuration">
          <menuitem type="checkbox"
            label="&index.menu_configuration_identification;"
            checked="true" disabled="true" />
        </menupopup>
      </menu>
    </menubar>
  </toolbox>
  ...
```

Le reste du menu *Configuration* est placé dans l'overlay. Si vous essayez l'exemple, vous remarquerez que l'application ne se comporte pas comme prévu : en effet, le menu *Configuration*, le premier à être mentionné dans le fichier appelant, est le premier à être positionné dans les menus. Or nous voulons qu'il soit placé en troisième position. Il faut donc écrire, dans l'overlay :

```
<menu id="xf-menu-fichier" label="&index.menu_fichier;"
  position="1">
  ...
<menu label="&index.menu_connexion;" position="2">
  ...
<menu label="&index.menu_configuration;"
  id="xf-menu-configuration" position="3">
```

Ainsi, avec les attributs `position`, on choisit une position dans un index numéroté à partir de 1. Dans les pages futures, nous n'aurons plus à nous préoccuper de l'ordre attribué aux différents menus, puisqu'il a été choisi de manière fixe dans l'overlay.

Par contre, dans le menu *Configuration*, l'élément que nous avons laissé dans `xulforum.xul` est placé en premier et non pas en dernier comme nous aimerions. Nous allons utiliser une autre technique pour forcer son positionnement en dernier. Dans l'overlay, nous allons préciser que nous voulons que les éléments du menu configuration soient placés avant l'élément « S'identifier auprès du serveur LDAP » du fichier appelant, d'attribut ID « `xf-menu-configuration-identification` ».

```
<menu label="&index.menu_configuration_appel;"
insertbefore="xf-menu-configuration-identification">...</menu>
<menuitem label="&index.menu_configuration_menus;"
insertbefore="xf-menu-configuration-identification"/>
```

Ainsi, nous obtenons le résultat voulu : notre élément est placé en dernier, car les éléments de l'overlay ont été insérés avant cet élément.

La barre d'outils

Nous allons maintenant créer la barre d'outils, autre élément inévitable dans une application graphique. Nous y placerons les quelques boutons indispensables : *Nouveau* pour un nouveau sujet, *Éditer*, *Supprimer* qui concernent un sujet sélectionné, *Plier*, *Déplier* concernent les pop-ups pour les messages et *Haut* qui permet de masquer tous les pop-ups.

Une barre d'outils (ou *toolbar* en anglais) se compose des éléments suivants : `<toolbar>`, un conteneur horizontal, qui contient des boutons spéciaux, `<toolbarbutton>`. Pour l'instant, nos boutons seront statiques ; ils n'auront pas d'effets spéciaux selon qu'ils sont enfoncés ou survolés par la souris, ou encore en-dehors de toute action spéciale. Ceci se fera avec des CSS, au chapitre suivant.

```
<toolbar id="xf-index-toolbar">
<toolbarbutton id="index-toolbar-nouveau"
  image="chrome://xulforum/content/images/new.png"
  tooltip="&index.toolbar_nouveau;" />
<toolbarbutton id="index-toolbar-editer"
  image="chrome://xulforum/content/images/edit.png"
  tooltip="&index.toolbar_editer;" />
<toolbarbutton id="index-toolbar-supprimer"
  image="chrome://xulforum/content/images/delete.png"
  tooltip="&index.toolbar_supprimer;" />
<toolbarbutton id="index-toolbar-plier"
  image="chrome://xulforum/content/images/1leftarrow.png"
  tooltip="&index.toolbar_plier;" />
<toolbarbutton id="index-toolbar-deplier"
  image="chrome://xulforum/content/images/1rightarrow.png"
  tooltip="&index.toolbar_deplier;" />
```

ASTUCE **Insertbefore, insertafter**

Vous pouvez contrôler encore plus finement le positionnement des éléments XUL de menu avec l'attribut *insertafter*, qui permet de préciser après quels éléments il faut insérer l'élément de l'overlay. Ceci n'est pas spécifique aux menus : vous pouvez également appliquer ce procédé au contenu d'une boîte par exemple.

OUTILS **Icônes, graphiques**

Nous utilisons pour notre application le *kit* graphique « lila », avec les icônes au format PNG, que l'on peut trouver sur le site <http://lila-theme.uni.cc/>. Placées sous licence GPL, ces icônes s'intègrent parfaitement dans notre application, elle aussi en GPL. La taille des icônes de la barre d'outils est de 32×32 et celle des menus de 16×16 . Ce *kit* est assez complet et propose des icônes pour la plupart des actions courantes, ce qui nous permet de trouver quasiment toutes les icônes nécessaires à l'application XUL Forum.

ASTUCE Personnalisation des barres d'outils

Après avoir lu le chapitre sur l'intégration dans Mozilla, il vous sera possible d'ajouter vos propres boutons aux barres d'outils du navigateur. Si vous utilisez Firefox, vous pourrez profiter de la fenêtre « Customize toolbars » avec l'élégant glisser/déposer pour organiser vos boutons. Ceci se fera dans le chapitre sur l'intégration dans Mozilla, quand nous rajouterons un bouton à la barre d'outils Firefox pour lancer directement XUL Forum.

```
<toolbarbutton id="index-toolbar-index"
  image="chrome://xulforum/content/images/luparrow.png"
  tooltiptext="&index.toolbar_index;" />
</toolbar>
```

L'élément `toolbar` fait toujours partie de l'overlay `index-barres-overlay.xul`. Nous ajoutons des éléments `<toolbarbutton>`, en spécifiant deux attributs : `image` et `tooltiptext`. Comme nous ne spécifions pas de texte (l'usage veut que les boutons, comme ceux du navigateur, soient uniquement des images), nous spécifions le texte d'infobulle, c'est-à-dire le texte qui apparaît quand l'utilisateur laisse sa souris pendant un court moment au-dessus du bouton.

Vous pouvez essayer d'ajouter un attribut `label`, mais le résultat n'est pas très esthétique, alors qu'avec les boutons, le haut de l'application commence déjà à faire professionnel !

La barre de statut

Enfin, dernier élément situé le plus en bas de l'écran, la barre de statut. Une barre de statut est comme une boîte horizontale normale, mais elle contient des éléments spéciaux `<statusbarpanel>`, qui sont responsables de la séparation nette entre les différents éléments de la barre de statut. Ce sont des boutons spéciaux, qui ne peuvent afficher qu'une image ou qu'un texte (et pas les deux à la fois).

Notre barre de statut contiendra au minimum deux éléments :

- un texte qui affichera ce que fait l'application : connexion au serveur, vérification de l'identité, etc.
- une barre d'avancement lors des connexions distantes, représentée par l'élément `<progressmeter>`.

```
<statusbar id="xf-index-statusbar">
  <statusbarpanel label="Bienvenue sur XUL Forum" />
  <statusbarpanel flex="1" />
  <statusbarpanel label="Non lus : 3" />
  <statusbarpanel label="Total : 5" />
  <statusbarpanel>
    <progressmeter />
  </statusbarpanel>
</statusbar>
```

La barre de statut est moins complexe que la barre d'outils ou la barre de menu : le texte et la barre d'avancement seront tous les deux manipulés par JavaScript, nous n'avons pas besoin de nous en occuper pour l'instant.

Nous voulons que le texte de statut soit placé à gauche et la barre d'avancement à droite de l'écran, avec l'indication du nombre de messages non lus et du nombre total de messages : nous rajoutons entre les deux éléments un `<statusbarpanel>` vide, avec un attribut `flex` à 1. Comme les autres n'ont pas d'attribut `flex`, ils prennent uniquement la place qui leur est nécessaire, tandis que le `<statusbarpanel>` qui a l'attribut `flex` prend toute la place qui lui est offerte et permet de coller chacun des éléments à droite ou à gauche de l'écran.

Les cases pour afficher le nombre de messages sont empruntées à Thunderbird : on retrouve exactement la même structure dans le coin inférieur droit du navigateur.

En ce qui concerne l'internationalisation de la barre de statut, nous n'avons pas à nous en préoccuper pour l'instant non plus. Comme elle sera entièrement manipulée par JavaScript, nous verrons dans le chapitre qui y est consacré la réponse à la question : « comment gérer l'internationalisation avec JavaScript » et sa solution interne.

Résumé : les barres

Nous avons vu l'utilisation de trois types de barres :

- la barre de menu, avec l'imbrication des pop-ups,
- la barre d'outils, avec les boutons spéciaux,
- la barre de statut, avec un autre type de bouton-conteneur.

Il nous faut maintenant construire la partie centrale de l'interface, avec l'arbre des messages et la liste des membres.

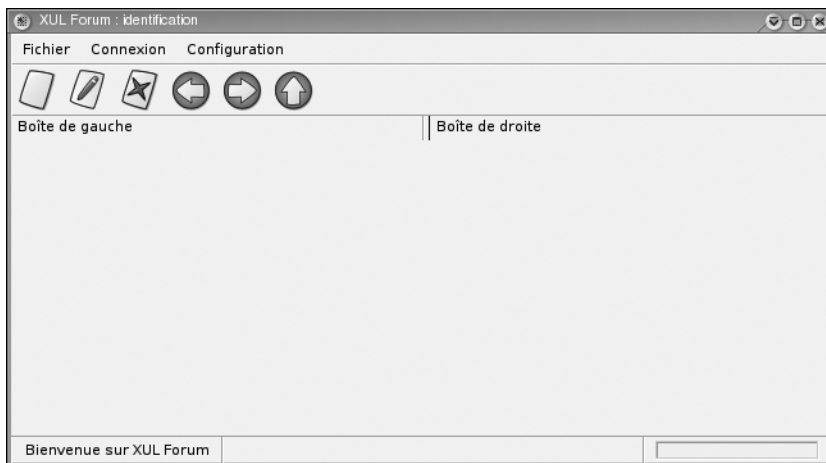


Figure 5–4
L'aspect actuel de la fenêtre de XUL Forum

Les overlays : arbre et onglets

Si vous testez le code dans son état actuel, vous remarquerez un petit problème : chaque élément de la fenêtre prend l'espace qui lui est nécessaire et ne s'étend pas en hauteur (non visible sur la figure précédente, pour épargner au lecteur des émotions fortes !). La barre de statut se retrouve ainsi à mi-hauteur de l'écran. Nous avons, d'après notre croquis, décrété que la boîte horizontale du milieu devait prendre toute la hauteur possible : nous allons reprendre le code de la fenêtre principale et rajouter des attributs pour mieux organiser l'occupation de l'espace.

```
...
<toolbox id="xf-index-toolbox" />
  <hbox flex="1">
    <vbox id="xf-index-vbox-membres">
      <description>Vbox de gauche</description>
    </vbox>
    <splitter><grippy /></splitter>
    <vbox id="xf-index-vbox-forum" flex="1">
      <description>Vbox de droite</description>
    </vbox>
  </hbox>
  <statusbar id="xf-index-statusbar" />
...
```

Comme la boîte horizontale a un attribut `flex`, elle occupe tout l'espace possible et « colle » la barre de statut en bas de la fenêtre. Ensuite, à l'intérieur de cette boîte horizontale, on met un attribut `flex` pour que la boîte de droite prenne tout la largeur possible. Sans ces précautions, nous nous retrouverions avec la barre de statut vers le milieu de l'écran et les deux boîtes verticales serait collées à gauche de l'écran, ce qui n'est pas très esthétique. Les attributs `flex` permettent donc de forcer des éléments à occuper tout l'espace possible, ce qui est plus seyant, notre fenêtre étant amenée à subir des redimensionnements multiples. Ainsi, quelle que soit la taille de la fenêtre, l'espace sera toujours occupé au maximum. Si on avait mis un attribut `flex` sur la boîte de gauche, elle aurait eu une largeur trop grande pour les onglets et le résultat n'aurait pas non plus été très joli, avec un grand espace vide entre les onglets et la poignée du milieu. Vous pouvez là encore jouer sur les attributs `flex` pour tester différents résultats.

Une fois remplie la partie de gauche concernant les membres, celle-ci aura une largeur minimum, du fait de ses onglets, et nous pourrons garder tout l'espace restant pour le forum lui-même.

Les étapes préliminaires étant maintenant achevées, nous pouvons remplir chacune de ces boîtes.

L'arbre

L'idée est ici d'écrire un arbre d'exemple, pour voir ensuite comment il sera possible de le manipuler à l'aide de JavaScript. Nous n'irons pas au-delà d'un seul niveau de profondeur, pour ne pas nous perdre dans l'imbrication des balises, déjà assez importante avec un seul niveau.

```
<?xml version="1.0" ?>
<!DOCTYPE overlay SYSTEM "chrome://xulforum/locale/xulforum.dtd">
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <vbox id="xf-index-vbox-forum">
    <label value="La liste des sujets du forum" />
    <tree id="xf-index-arbre" flex="1">
      <treecols>
        <treecol id="xf-index-arbre-titre" label="Titre du sujet" flex="8" />
        <splitter class="tree-splitter" />
        <treecol id="xf-index-arbre-date" label="Dernier message le :" flex="1"/>
        <splitter class="tree-splitter" />
        <treecol id="xf-index-arbre-reponses" label="Nombre de réponses" flex="1"/>
      </treecols>
      <treechildren>
        <treeitem>
          <treerow>
            <treecell label="Un exemple de titre de sujet" />
            <treecell label="20/02/05 16:30" />
            <treecell label="1" /><!-- on retrouve trois éléments car on a spécifié
                                trois colonnes, comme dans le <grid> déjà vu -->
          </treerow>
        </treeitem>
      </treechildren>
    </tree>
  </vbox>
</overlay>
```

Voici le fichier `index-forum-overlay.xul`, mentionné lorsque nous avons écrit le squelette principal de la page.

ALTERNATIVE Un arbre HTML ?

Pour l'instant, tout ce que nous avons vu est réalisable en XHTML, même s'il y aurait quelques difficultés à rendre une barre d'outils « propre ». Les menus seraient réalisables avec des CSS judicieusement utilisées, comme l'illustre le site d'Éric Meyer.

► <http://meyerweb.com/eric/css/edge/menus/demo.html>

On pourrait également penser aux *frames* du HTML pour le découpage de l'interface en plusieurs éléments, ou même aux tableaux, pour tout ce qui concerne la répartition de l'espace entre les différents éléments. Mais maintenant, pour un arbre hiérarchique, XUL va vraiment démontrer sa supériorité par rapport à un client de type léger. Un arbre n'est pas plus complexe que les autres éléments, alors qu'en HTML, il n'est certainement pas possible d'implémenter simplement un arbre de ce type !

- http://www.xulplanet.com/references/elemref/ref_treecol.html
- http://www.xulplanet.com/references/elemref/ref_tree.html

Il contient principalement un élément `tree`, avec un attribut `flex` à 1, pour qu'il occupe toute la hauteur de la boîte verticale dans laquelle il est placé. Chaque colonne est annoncée sous la forme d'un élément `<treecol>` placé dans un conteneur `<treecols>`. Les `splitters`, déjà vus précédemment, possèdent cette fois une classe spéciale qui leur permet d'adopter une largeur de 1 pixel et de permettre le redimensionnement des colonnes du tableau. Les attributs `flex` servent à répartir l'espace entre les colonnes. Une dernière colonne est ajoutée automatiquement : c'est le *column picker*, qui permet de choisir quelles colonnes afficher ou masquer. Vous pourrez alors utiliser des attributs comme `hidden` pour ne pas les afficher par défaut, ou `ignoreincolumnpicker` pour ne pas les afficher dans le sélecteur de colonnes. La liste complète des attributs de la balise `<treecol>` se trouve sur le site de [xulplanet.com](http://www.xulplanet.com).

L'élément `tree` possède aussi des attributs intéressants, comme `alternatingbackground`, pour alterner la couleur des lignes, ou `enableColumnDrag`, pour permettre à l'utilisateur de déplacer les colonnes et réagencer leur succession avec un glisser/déposer.

Voici maintenant l'exemple revu, avec un nouveau niveau de profondeur, pour les réponses à un sujet.

index-forum-overlay.xul, avec un nouveau niveau de profondeur

```
<?xml version="1.0" ?>
<!DOCTYPE overlay SYSTEM "chrome://xulforum/locale/
xulforum.dtd">
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/
there.is.only.xul">
  <vbox id="xf-index-vbox-forum">
    <label value="&index.arbre_label_titre;" />
    <tree id="xf-index-arbre" flex="1">
      <treecols>
        <treecol id="xf-index-arbre-titre"
          label="&index.arbre_titre;" flex="8" primary="true"/>
        <splitter class="tree-splitter" />
        <treecol id="xf-index-arbre-date"
          label="&index.arbre_date;" flex="1"/>
        <splitter class="tree-splitter" />
        <treecol id="xf-index-arbre-reponses"
          label="&index.arbre_reponses;" flex="1"/>
      </treecols>
      <treechildren>
        <treeitem container="true">
          <treerow>
            <treecell label="Un exemple de titre de sujet" />
            <treecell label="20/02/05 16:30" />
            <treecell label="1" />
          </treerow>
          <treechildren>
            <treeitem>
```

```

        <treerow>
          <treecell label="Réponse au premier sujet" />
          <treecell label="21/02/05 00:05" />
          <treecell />
        </treerow>
      </treeitem>
    </treechildren>
  </treeitem>
</treechildren>
</tree>
</vbox>
</overlay>

```

Plusieurs choses ont été modifiées. La première colonne a maintenant un attribut `primary`, qui indique qu'elle se destine à contenir des éléments imbriqués et que c'est dans cette colonne que devra être affiché le petit + ou - selon que l'élément est déplié ou non. Le `<treeitem>` a son attribut `container` à `true`, puisqu'il se destine à contenir d'autres éléments. En effet, à la suite du `<treerow>`, on réitère le procédé d'ajout d'une ligne, mais cette fois à un nouveau niveau de profondeur, avec un nouveau `<treechildren>`. Le résultat peut être visualisé, il est fonctionnel. Cet arbre n'est bien sûr qu'un exemple, il sera destiné à être manipulé avec JavaScript et RDF un peu plus loin.

La liste des membres

La liste des membres, dernière grande partie de l'interface, se composera de deux onglets, l'un pour la liste des membres, l'autre pour voir des informations sur un membre particulier. Ce sera le fichier `index-membres-overlay.xul`.

```

<?xml version="1.0" ?>
<!DOCTYPE overlay SYSTEM "chrome://xulforum/locale/xulforum.dtd">
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  there.is.only.xul">
  <vbox id="xf-index-vbox-membres">
    <label value="Les membres du forum" />
    <tabbox flex="1">
      <tabs>
        <tab label="Liste des membres" />
        <tab label="Infos sur le membre" />
      </tabs>
      <tabpanel flex="1">
        <tabpanel>
          <listbox>
            <listitem label="Jonathan" />
            <listitem label="Stéphane" />
            <listitem label="Muriel" />
            <!-- etc. tout cela sera modifié avec JS -->
          </listbox>

```

```

        </tabpanel>
        <tabpanel>
            <vbox>
                <hbox>
                    <image src="chrome://xulforum/content/01.png"
                        width="32" height="32"/>
                    <description flex="1">Nom du personnage
                </description>
            </hbox>
            <label value="Âge: 18 ans" />
            <label value="Localisation: Bordeaux" />
            <label value="Posts: 55" />
        </vbox>
    </tabpanel>
</tabpanel>
</tabpanels>
</tabbox>
</vbox>
</overlay>

```

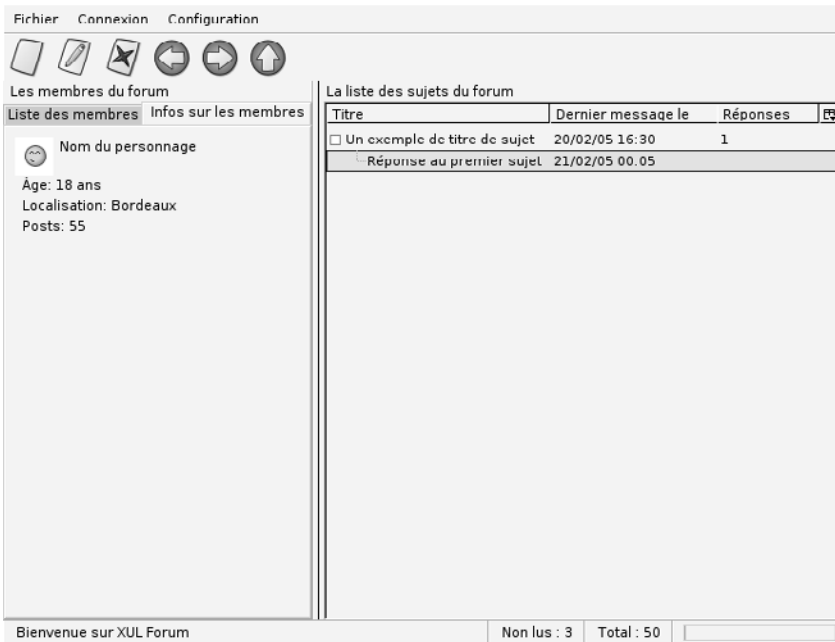
L'utilisation d'un panneau à onglets ressemble fortement à celle d'un arbre : d'abord la liste des onglets, puis la liste des panneaux. Les attributs `flex` placés sur `tabpanel`s et sur `tabbox` obligent les panneaux à prendre toute la hauteur disponible dans l'écran.

Chaque élément `tab` annonce un panneau et représente de manière concrète l'onglet en lui-même, c'est-à-dire le petit conteneur de forme arrondie dans les angles, contenant le titre de chaque panneau. À chaque élément `tab` doit correspondre un élément `tabpanel`.

Ensuite, les `tabpanel`s sont placés. Il est possible de remplacer le `tabpanel` par un élément unique, mais ceci n'est pas recommandé. L'élément `tabpanel` est encore un conteneur vertical spécial, dont l'orientation se choisit grâce à l'attribut `orient`. Les panneaux peuvent être placés sur la gauche, en bas, etc. La référence complète est une fois de plus la meilleure amie du développeur !

Dans le premier panneau, on insère une liste des personnes présentes via l'élément `<listbox>` : à chaque entrée dans la liste correspond un `<listitem>`. Cet élément sera manipulé avec JavaScript.

Ensuite, dans le deuxième panneau, on prépare l'écran qui sera construit dynamiquement lorsque l'utilisateur demandera à voir des informations sur un membre en particulier. On y trouve la photo de l'utilisateur (une photo d'exemple est utilisée pour voir le résultat), quelques informations le concernant, ainsi que sa description. L'idée est surtout de s'arranger pour voir comment gérer les attributs `flex`, la répartition de l'espace et le positionnement des différents textes.

**Figure 5–5**

Le résultat final sous Linux avec KDE, Firefox et le thème Lila.

Vous remarquerez que l'on a ajouté des titres aux deux parties, membres et arbre.

En résumé...

Nous ne nous sommes pas attardés sur le remplissage des éléments du milieu de l'interface : il faut se dire que nous posons juste les bases et que tout cela sera manipulé avec JavaScript ou rempli automatiquement avec RDF. Par contre, les menus et les barres d'outils sont des éléments plus importants qu'il faut répartir soigneusement entre plusieurs fichiers, afin de pouvoir par la suite les incorporer facilement dans n'importe quelle page.

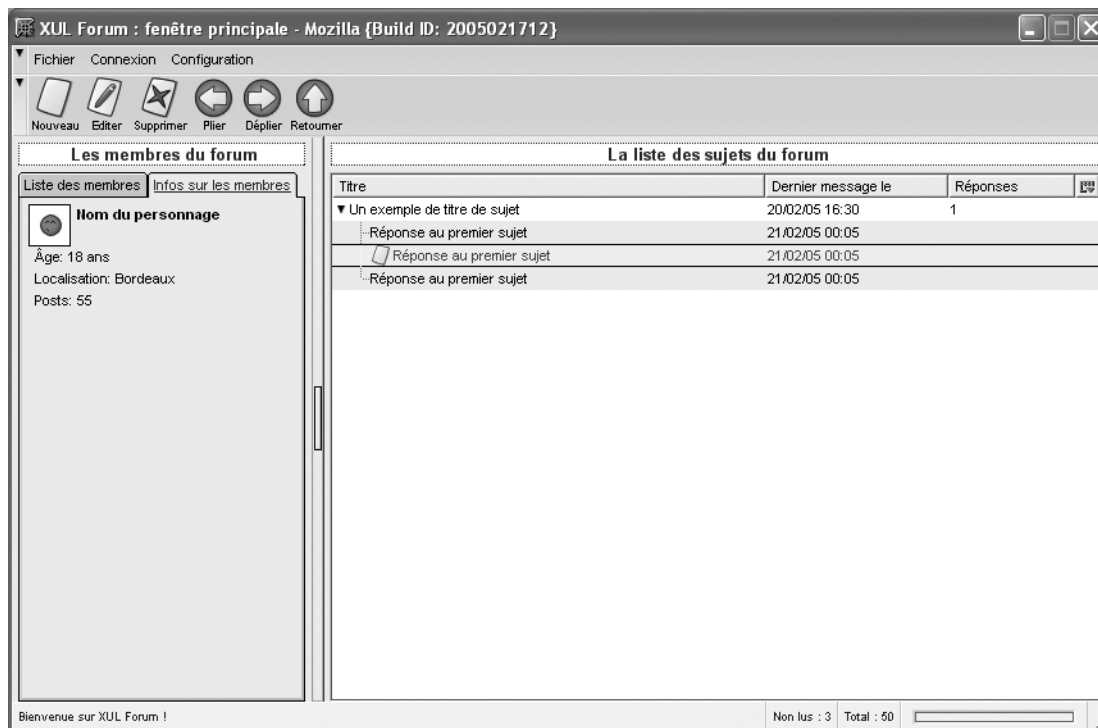
L'interface, en version statique, de XUL Forum, est maintenant quasiment définitive. Nous avons successivement codé :

- les barres d'outils : menu, boutons, barre de statut,
- le découpage en overlays du menu, en séparant les éléments « globaux » des éléments spécifiques à chaque page,
- les boîtes du milieu de l'écran, avec des ébauches d'arbre et de liste des membres, qui seront manipulées plus tard avec JavaScript.

Quel est le travail restant sur l'interface ? Surtout l'intégration des graphiques, avec des effets sur les boutons au passage de la souris, des icônes pour certaines options de menu, mais aussi la mise en forme : mise en relief des titres, choix des polices, des effets sur les polices... bref, de la présentation par des CSS !

6

chapitre



Perfectionnement du rendu avec CSS

L'interface commence à être finalisée mais n'est pas encore parfaite : il lui manque la « touche finale » en termes de rendu, que nous réaliserons grâce à des CSS.

SOMMAIRE

- ▶ Effets simples : mise en forme de texte
- ▶ Retouches structurelles : modifications des marges et bordures
- ▶ Propriétés propres à Mozilla : icônes des menus, boutons de la barre d'outils

MOTS-CLÉS

- ▶ Sélecteurs CSS
- ▶ Propriété list-style-image
- ▶ -moz-xxxx

XUL Forum ressemble maintenant à quelque chose. L'interface ne bougera quasiment plus dans sa structure, les widgets XUL sont choisis et positionnés. Cependant, elle garde quelques défauts : des titres pourraient être mis plus en relief, des éléments ont une bordure trop importante, les boutons de la barre d'outils ne sont pas aussi bien réalisés que ceux de la fenêtre du navigateur... une retouche de l'ensemble de l'application avec CSS s'impose !

Présentation de CSS ; utilisation dans Mozilla

CSS, ou *Cascading Style Sheets*, se décline en plusieurs versions. La première version, CSS1, a été adoptée comme recommandation du W3C en 1996. Son but était de proposer un langage normalisé visant à définir le style des pages web et de mettre fin aux styles propriétaires propres à chaque type de navigateur. La particularité de ce projet était de proposer un modèle en cascade : le style spécifié par l'auteur pouvait être modifié par les préférences de l'utilisateur. Ce concept de cascade se retrouve plus en détail lorsque l'on crée une feuille de style : selon que le style est placé directement dans le document, qu'il concerne un groupe d'éléments ou un élément particulier, il n'a pas la même priorité et, par conséquent, pas la même hauteur dans la cascade ! (Voir le lien sur l'excellent article d'OpenWeb ci-contre.) Les implémentations correctes de CSS1 virent le jour en 2000 avec Microsoft Internet Explorer pour Mac OS, mais aussi des navigateurs alternatifs comme Opera ou, bien sûr, Mozilla.

La seconde version de CSS fut adoptée comme recommandation en 1998. Cependant cette version, peut-être trop ambitieuse, n'est toujours pas entièrement implémentée, et ce dans quelque navigateur que ce soit. CSS 2.1, sorte de version revue et corrigée, a élagué toutes les spécifications qui n'avaient pas été implémentées. À ce jour, la prochaine version CSS 3 est toujours en cours de développement.

La norme CSS, dans le monde du Web, est utilisée principalement pour permettre la séparation du contenu logique (écrit en HTML) et de l'apparence. Lorsqu'on crée une page web avec CSS, le document HTML est structurel : des balises comme h1 ou strong prennent tout leur sens : « titre principal » pour h1, « texte à mettre en relief » pour strong. La feuille de style, séparée du document HTML, précise comment rendre graphiquement cette structure du document. Souvent, plusieurs feuilles de style sont disponibles, offrant un choix à l'utilisateur. Malheureusement, cette utilisation idéale de CSS n'est le plus souvent une réalité que sur les sites créés par les ardents défenseurs de l'accessibilité ! Le reste des développeurs web ne s'en servent finalement

► http://openweb.eu.org/articles/cascade_css/

CULTURE CSS Zen Garden

CSS Zen Garden est un site « vitrine » du CSS. En variant indéfiniment la feuille de style d'un même document HTML, il montre toutes les possibilités de CSS et l'importance de la feuille de style dans le rendu d'un document bien présenté. À partir d'un même document HTML, des feuilles de style changeant radicalement la conception du site.

- <http://www.csszengarden.com/?cssfile=http://www.css-praxis.de/cssocean/zenocean.css>
 - <http://www.csszengarden.com>
 - <http://csszengarden.com/?cssfile=http://www.sprae.com/csszen/EmptinessIsZen.css>
-

que pour gagner du temps dans la rédaction du HTML. Il ne faut d'ailleurs pas ignorer les nombreux bogues d'implémentation de CSS dans les navigateurs, qui freinent encore son adoption massive. Citons à ce titre le célèbre bogue du modèle de boîte de Microsoft Internet Explorer, qui oblige les développeurs web à exploiter les conséquences imprévues de ce problème pour rendre correctement leur document sur tous les navigateurs !

Dans Mozilla, CSS est utilisé un peu différemment. Bien sûr, toutes les possibilités offertes dans un contexte web se trouvent implémentées. On aura donc toujours l'interface structurale, décrite en XUL et la présentation graphique, écrite en CSS. Mais Mozilla propose également ses propres extensions complémentaires à CSS. Celles-ci vont de simples effets visuels (tels que des bordures arrondies ou la gestion de la transparence) à des propriétés plus complexes comme l'attribution pour un widget du fichier XBL lui correspondant. Nous allons d'abord voir les propriétés « classiques » de CSS, puis ensuite les propriétés propres à Mozilla.

► http://en.wikipedia.org/wiki/Internet_Explorer_box_model_bug

CULTURE « Cutting edge CSS »

Voici l'adresse de la célèbre *Complex Spiral Demo*, créée par Eric Meyer, un auteur reconnu dans le domaine des CSS. Elle montre, si votre navigateur est compatible, ce qu'il est possible d'accomplir en utilisant les seules CSS, en une éblouissante prouesse technique. Sous Mozilla, il ne devrait pas y avoir de problème !

► <http://meyerweb.com/eric/css/edge/complexspiral/demo.html>

ATTENTION CSS, un must du développement web

Lorsque l'on vient d'un environnement web qui laisse une large part à la programmation (comme les habitués de PHP/MySQL), on a tendance à considérer CSS comme un gadget, une alternative sympathique mais inutile face à une mise en forme en HTML *quick & dirty* (vite et pas très propre). Il faut cependant garder à l'esprit ses nombreux avantages pour les sites web, dont voici quelques exemples :

- **Bande passante économisée.** En centralisant la mise en forme dans un seul fichier CSS, les navigateurs mettront la feuille de style dans leur cache local une bonne fois pour toutes, pour l'ensemble des pages consultées.
- **Séparation totale de la présentation et du contenu.** Les navigateurs en mode texte ou à destination des mal ou non-voyants ou encore les anciens navigateurs afficheront le contenu de manière claire. Les moteurs de recherche eux aussi « comprendront » mieux votre site, lui donnant sans doute une meilleure visibilité ! Par ailleurs, les nouveaux vecteurs de communications (comme les

PDA ou les téléphones portables) auront une mise en page correcte du contenu de votre site.

- **Modularité accrue.** Au moment où vous aurez à changer les polices de caractère utilisées dans votre site, vous n'aurez qu'un attribut CSS à modifier !
- **Compatibilité.** Standardisées et complètes, les CSS limitent la nécessité d'extensions propriétaires à l'origine de la « balkanisation » du Web.

Nous vous encourageons fortement à utiliser les standards pour créer votre site web – en vous gardant toutefois de verser dans l'intégrisme : il n'est toujours pas possible de concevoir un site uniquement en XHTML et CSS : ce serait mettre de côté le rendu des navigateurs d'ancienne génération. Le livre de Jeffrey Zeldman est une référence en la matière : l'auteur propose des mises en page hybrides, utilisant XHTML et CSS sans pour autant laisser de côté des navigateurs tels que Netscape 4 ou Internet Explorer 4.

📖 *Design web - Utiliser les standards*, Jeffrey Zeldman, Eyrolles

Débuts avec CSS : effets sur du texte

Mise en place de CSS

Comme à notre habitude, nous allons commencer par l'écran d'authentification : il fera un bon point de départ pour nos premiers effets CSS. Le premier problème qu'il convient de résoudre est le placement des styles. Où les ranger ? Directement dans le document XUL ? Dans une feuille de style à part, comme en HTML ? Les intégrer en tant que feuille de style XML ?

La première solution est à éliminer tout de suite, car le but de CSS est justement de séparer la structure de l'apparence. Le développeur doit pouvoir facilement changer de feuille de style pour son application XUL sans avoir à modifier les fichiers XUL dès qu'il veut changer l'aspect de XUL Forum. De plus, ces styles ont une priorité importante qui fait que, généralement, ils écrasent les styles définis dans le thème Mozilla. Les styles `inline`, c'est-à-dire contenus dans l'attribut `style` d'une balise XUL, sont donc à éviter, de même que les balises `<style>`.

La seconde solution utiliserait une déclaration de type :

```
<link rel="stylesheet" ...>
```

bien connue dans le monde du HTML. Mais pourquoi utiliser une balise HTML alors que Mozilla et ses fichiers XML conformes permettent d'utiliser la syntaxe XML pour appliquer une feuille de style ?

Nous utiliserons donc la syntaxe la plus adaptée à nos fichiers XUL, qui sont avant tout des fichiers XML : c'est la ligne de type `<?xml-stylesheet... ?>` que nous ajouterons dans le prologue XML de `xulforum.xul`.

```
<?xml-stylesheet href="chrome://xulforum/skin" type="text/css" ?>
```

Cette ligne remplace la ligne précédente qui importe le thème global de Mozilla. Mais le thème global n'est pas oublié pour autant. Il sera simplement importé depuis le skin de XUL Forum pour plus de simplicité.

ALTERNATIVES CSS dans content ?

Si vous disséquez les extensions Firefox disponibles, vous remarquerez que toutes ne sont pas respectueuses de la structure tripartite d'une extension et qu'elles placent leurs fichiers CSS dans le dossier `content`. Ceci permet un gain de temps appréciable, car il n'y a pas à avoir un autre fichier `contents.rdf`, ni à créer un autre dossier. Cependant, si vous voulez proposer une extension à XUL Forum, comme un thème assorti à Firefox, il faut absolument respecter cette structure qui garantit une large modularité.

Vous remarquerez qu'une fois de plus, le fichier est placé dans une URL de type `chrome : xulforum.css` sera dans le dossier `skin` de l'application. Il est donc nécessaire de créer le fichier `contents.rdf` du dossier `skin`, afin de pouvoir enregistrer l'habillage de XUL Forum auprès du navigateur.

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
  <RDF:Seq about="urn:mozilla:skin:root">
    <RDF:li resource="urn:mozilla:skin:classic/1.0" />
  </RDF:Seq>
  <RDF:Description about="urn:mozilla:skin:classic/1.0">
    <chrome:packages>
      <RDF:Seq about="urn:mozilla:skin:classic/1.0:packages">
        <RDF:li resource="urn:mozilla:skin:classic/1.0:xulforum"/>
      </RDF:Seq>
    </chrome:packages>
  </RDF:Description>
</RDF:RDF>
```

Ce fichier est très similaire à celui définissant la locale `fr-FR`, vue précédemment. D'abord, on précise que ce qui va suivre concerne l'habillage *classic/1.0*, nom utilisé en interne pour désigner l'habillage classique de Mozilla. Ensuite, on remplit l'élément `Description` se référant au skin classique. On indique que l'on va y ajouter un nouveau paquet, en l'occurrence un paquet concernant XUL Forum.

Nous ajoutons donc au skin classique le paquet correspondant au composant XUL Forum. Mais attention, il ne s'agit pas de redéfinir *toutes* les propriétés CSS du thème mais simplement de remplacer certaines propriétés globales par celles de notre application ; c'est pourquoi nous commencerons notre fichier `xulforum.css` par cette ligne :

```
@import url("chrome://global/skin");
```

Cette ligne indique que nous intégrons d'abord l'ensemble des propriétés CSS définies par le skin en cours du navigateur, pour ensuite apporter nos propres modifications. Ainsi, nos propriétés CSS auront priorité par rapport aux propriétés globales, puisqu'elles seront définies plus haut dans la cascade CSS.

CULTURE Effet des CSS sur une application XUL

Nous avons intégré la feuille de style globale du navigateur dès le premier chapitre d'écriture de XUL. Ainsi, notre application a immédiatement été intégrée dans le navigateur. Si vous voulez voir quel est l'effet des CSS sur XUL Forum, supprimez la directive d'import du skin global dans `xulforum.css`. Vous verrez alors le fichier XUL « à nu », sans sa CSS. La structure du fichier est bien là, mais toute la présentation a disparu ! En fait, nous avons déjà profité largement des CSS mais sans trop nous en rendre compte.

COMPRENDRE Les habillages (skins) dans Mozilla

L'idée importante à saisir est celle du concept d'habillage ou *skin* dans Mozilla. Un habillage est un ensemble de paquets concernant chacun une partie de l'interface Mozilla. À la base, se situe l'habillage classique, qui contient par exemple des paquets définissant l'habillage global, celui du navigateur, du client de messagerie, etc. Peut s'y ajouter un habillage personnalisé, si l'utilisateur le désire. Ce peut être le skin fourni appelé *modern*, ou un autre disponible sur le site `update.mozilla.org`. Les propriétés spécifiées dans le fichier de skin de l'utilisateur écrasent celles présentes au départ dans l'habillage classique.

Comme pour le dossier *content*, le fichier par défaut utilisé pour l'URL `chrome://xulforum/skin/` est `xulforum.css`. Ceci permet d'utiliser un raccourci dans les prologues XML.

Même si notre skin est défini comme appartenant par défaut au skin classique, il sera tout de même choisi si le skin de l'utilisateur est différent, puisqu'il n'y a pas d'autre choix. On peut en revanche imaginer que, lorsque XUL Forum sera internationalement connu, les concepteurs de skin intégreront des skins pour XUL Forum, qui pourront écraser ce que nous avons défini dans le skin classique pour XUL Forum.

Il faut maintenant finaliser l'installation du skin : rajoutez une ligne au fichier `installed-chrome.txt` :

```
skin,install,url,file:///mnt/data/Livre/xulforum/skin/
```

Bien sûr, vous devrez adapter le chemin vers votre répertoire de travail. Le mieux est de vous fonder sur la ligne précédente du fichier, qui doit être celle de votre locale.

Supprimez `chrome.rdf`, n'oubliez pas de créer le fichier `xulforum.css` basique dans le dossier `skin`, relancez le navigateur et pointez-le sur `chrome://xulforum/content`. Il n'y aura aucun changement visible et ceci est parfaitement normal ! Nous avons juste créé l'habillage de base, mais nous n'y avons rien mis encore ! Désormais, les modifications que nous allons placer dans `xulforum.css` seront apparentes.

Premiers effets sur du texte

Maintenant nous pouvons commencer à agir concrètement sur l'apparence rendue du fichier `xulforum.xul`. La première chose qui s'impose est de mettre plus en relief le titre *Identification*. On ne voit pas assez, quand on arrive sur la page d'accueil de XUL Forum, que l'on demande à l'utilisateur de s'identifier. Pour cela, nous pouvons ajouter dans `xulforum.css` :

```
.titre {
    font-weight: bold;
    background-color: white;
}
```

C'est ici un sélecteur de classe qui est employé, afin de pouvoir centraliser des attributs communs à tous les éléments jouant le rôle de titre. En effet, l'élément titre est ici un `caption`. Cependant, dans la page suivante, `index.xul`, les éléments titres seront de type `label`. Les attributs précisés ici seront donc communs aux deux éléments `caption` et `label`, portant l'attribut `class="titre"`. Nous préciserons plus loin les attributs CSS spécifiques aux éléments titres `label`.

Rappel Annexe

Vous pourrez retrouver la liste des principales propriétés CSS en annexe D, ainsi qu'une remise en situation de sa syntaxe.

B.A.-BA Syntaxe CSS

Les fichiers CSS suivent une syntaxe fort simple. Elle ressemble plus ou moins à ceci :

```
sélecteur {
  attribut1: valeur1;
  attribut2: valeur2;
}
```

Les attributs seront introduits au fur et à mesure de ce chapitre, lorsque nous en aurons besoin. Pour une liste complète des attributs, une recherche sur Internet à « CSS Reference » vous donnera un bon nombre de résultats.

► http://www.w3schools.com/css/css_reference.asp

📖 R. Goetter, *CSS 2 – Pratique du design web*, Eyrolles 2005

📖 J. Zeldman, *Design web – Utiliser les standards*, Eyrolles 2005

📖 E. Meyer, *CSS – Précis et Concis*, O'Reilly 2004

📖 E. Meyer, *CSS par Éric Meyer*, CampusPress 2005

En ce qui concerne les sélecteurs, nous les découvrirons de même au fur et à mesure de ce chapitre. Voici les principaux :

- `label {}` pour toucher tous les éléments `label`.
- `label.titre {}` pour tous les `<label class="titre" .../>`.
- `label#premierTitre {}` pour l'unique `<label id="premierTitre" .../>`.
- `hbox label {}` pour tous les `<label>` inclus dans une `<hbox>`, un ou plusieurs niveaux de profondeur en-dessous de la balise `<hbox>`.
- `hbox > label {}` pour les `<label>` directement enfants du `<hbox>`, sans intermédiaire.
- `.titre` pour tous les éléments portant l'attribut `class="titre"`.
- `#titrePrincipal` pour tous les éléments portant l'attribut `id="titrePrincipal"`.

Une référence complète des sélecteurs CSS est également disponible en annexe.

Nous n'utilisons que deux attributs très simples : `font-weight`, qui précise le « poids », c'est-à-dire la graisse de la police : ici, `bold` pour une police en gras. D'autres valeurs existent, par exemple `bolder` ; des valeurs numériques sont également possibles, mais moins usitées. `background-color`, comme son nom l'indique, précise la couleur de fond, ici, le blanc.

On pourrait être tenté de choisir une autre couleur de fond que le blanc, pour être plus en accord avec le thème actuel. Ainsi, si vous avez Mozilla et le thème moderne, vous choisirez peut-être un bleu pour cette *background color*. Mais il faut garder à l'esprit un point important : l'utilisateur aura quasiment toujours un style différent de celui du développeur. C'est pourquoi, lorsque l'on écrase les réglages du thème principal, comme c'est le cas ici, il faut se garder d'utiliser des couleurs trop criardes. Ici, le blanc est assez neutre et s'accommodera de la plupart des thèmes imaginables.

B.A.-BA Couleurs en CSS

En CSS, il existe plusieurs moyens de spécifier une couleur. Pour les couleurs les plus courantes, une valeur littérale (`blue`, `red`, `white`, etc.) suffira. Pour plus de précision, on utilise souvent la syntaxe HTML `#012345` composée de trois nombres hexadécimaux à la suite. Notez qu'on peut abréger une couleur comme `#00CCFF` en `#0cf` (majuscules ou minuscules n'ont pas d'importance). Il existe enfin une dernière forme, peu courante : `rgb(0,0,255)` ou `rgb(0%,0%,100%)` par exemple.

Vous trouverez à l'adresse suivante un très bon outil qui utilise le CSS et permet d'ajuster facilement la couleur idéale :

► <http://meyerweb.com/eric/tools/color-blend/>

OUTILS WYSIWYG dynamique pour l'édition de CSS dans Mozilla Firefox

Si vous avez installé Mozilla avec la *Web Developer Toolbar* (disponible sous forme d'extension pour Firefox), en sélectionnant dans le menu *CSS View Style Information*, vous pourrez, en promenant votre souris sur l'écran, voir le sélecteur le plus détaillé qui correspond à l'élément sur lequel est placé votre souris. Souvent, toutes les balises parentes sont précisées, alors qu'en pratique, on utilise des sélecteurs plus simples, ne reprenant généralement que les deux derniers éléments de la liste.

ASTUCE « Sidebars » pour une référence

Si vous ne les connaissez pas encore, n'hésitez pas à installer les barres de DevEdge (l'ancien site des développeurs Netscape, fermé mais en cours d'intégration au site mozilla.org), pour avoir rapidement sous vos yeux la référence de tel ou tel élément CSS ou HTML. Ce type d'outil se révèle vite, à l'usage, indispensable ! Une fois ces barres installées, vous y accédez rapidement avec la touche F9 sous Mozilla.

L'équivalent existe pour les éléments XUL et se trouve sur le site xulfr.org

- <http://devedge-temp.mozilla.org/toolbox/sidebars/>

Il faut maintenant modifier le fichier XUL pour indiquer que l'élément `caption` qui contient le titre du cadre appartient bien à la classe `titre`.

```
<caption class="titre" label="&ident.titre;" />
```

Il suffit simplement d'ajouter cet attribut à la balise et d'actualiser la page (en supposant que vous avez désactivé la mise en cache XUL).

Le titre ressort alors de manière plus flagrante, ce qui est plus agréable à l'œil et plus lisible pour l'utilisateur ! Bien sûr cet effet est fort simple, mais il permet de mettre en évidence la méthode.

Nous sommes maintenant prêts à appliquer du style à un élément plus compliqué. Des variantes sont naturellement possibles dans le fichier CSS, pour nos éléments `label` de `index.xul`, indiquant *Les membres du forum* ou *La liste des sujets du forum*. Vous pouvez même, si vous le désirez, vous amuser avec les premiers effets sur ce titre : `text-transformation`, `font-style`, `text-decoration`, `border-color`, `border-width`, `border-style`, `font-family`, `color`, à vous de découvrir leurs différentes utilisations !

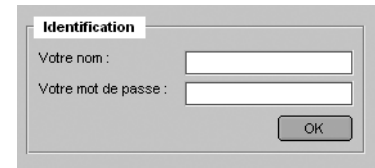


Figure 6-1

Maintenant, nous pouvons par exemple nous occuper des titres intermédiaires de l'écran principal. Voici la définition CSS qui leur correspond :

```
label.titre {
    border: 1px dotted navy;
    margin: 5px;
    padding-left: 5px;
    color: #036;
}
```

Pour les éléments `label` de la classe `titre`, nous ajoutons une bordure d'un pixel en pointillés. L'attribut `border` est un raccourci. Nous pourrions le détailler en :

```
label.titre {
    border-width: 1px;
    border-style: dotted;
    border-color: navy;
}
```

L'attribut `margin` signale qu'à l'extérieur de la bordure, il doit y avoir cinq pixels de distance avec les autres éléments. L'attribut `padding-left` indique quant à lui que, entre la bordure gauche et le début du texte, il doit y avoir également cinq pixels. Il existe bien sûr toutes les variantes pour chacun de ces deux attributs : `margin-left`, `margin-right`, `margin-top`, `margin-bottom`, `margin`, ainsi que `padding`, `padding-left`, etc.

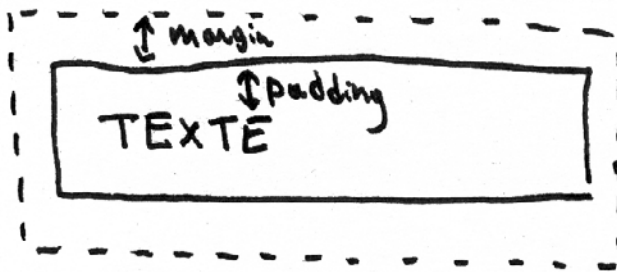


Figure 6–2

Représentation des grandeurs `margin` et `padding`. Le trait en pointillés représente la limite extérieure de l'élément. Le trait continu représente sa bordure, qui peut elle aussi avoir une largeur.

La couleur est quant à elle spécifiée sous la forme abrégée de `#003366`. C'est un bleu marine qui n'est pas trop agressif. On aurait aussi pu centrer le texte dans les `label`. Ceci se fait avec la propriété `text-align: center;`.

En ce qui concerne l'onglet *Informations sur un membre*, on peut là aussi effectuer quelques modifications pour en rendre la lecture plus agréable.

```
label.infosMembre:hover {
    text-decoration: underline;
}

description#nomPerso {
    font-weight: bold;
}

image#avatar {
    border: 1px solid black;
    width: 33px;
    height: 33px;
}
```

Le premier sélecteur utilise un pseudo-style, spécifié après les deux points. Un pseudo-style sert à définir un cas particulier pour un élément, par exemple lorsque cet élément est survolé par la souris (c'est le cas de `hover`), ou lorsque le lien a déjà été visité (attribut `visited` des liens HTML). La liste complète est disponible dans la référence CSS.

► <http://devedge.entangledesign.com/sidebars/css2.1/CSS21/selector.html#q15>

On souligne ainsi du texte comme « Ville: Bordeaux » lorsque l'utilisateur passe la souris dessus, pour lui en faciliter la lecture.

Pour l'élément `description` contenant le pseudonyme de l'utilisateur sélectionné, on se contente de le mettre en gras, pour le faire ressortir au milieu des informations secondaires de l'onglet.

Enfin, pour l'image, on spécifie ses dimensions dans le fichier CSS, ce qui rend une migration à des images de taille supérieure plus aisée. Les dimensions sont augmentées d'un pixel, pour prendre en compte le pixel de la bordure. La propriété `border` est elle aussi un raccourci. Voici sa forme développée :

```
border-width: 1px;
border-color: black;
border-style: solid;
```

Le tag de l'image de l'utilisateur dans le fichier XUL est donc fortement allégé. Par contre, il ne faut pas oublier d'ajouter les attributs `id` et `class` sur les différents éléments du panneau. Voici à quoi ressemble le fichier XUL maintenant :



Figure 6-3

```
<vbox>
  <hbox>
    <image src="chrome://xulforum/content/01.png" id="avatar"/>
    <description flex="1" id="nomPerso" value="Nom du personnage" />
  </hbox>
  <label class="infosMembre" value="Âge: 18 ans" />
  <label class="infosMembre" value="Localisation: Bordeaux" />
  <label class="infosMembre" value="Posts: 55" />
</vbox>
```

Enfin, si pour des questions de goût, la police utilisée par défaut ne vous plaît pas, vous pouvez choisir la vôtre. En CSS, nous utiliserons le sélecteur universel pour appliquer une police à tous les éléments.

```
* {
  font-family: Helvetica, Arial, Sans-Serif;
}
```

Il ne faut surtout pas oublier de terminer la liste des polices possibles par une police standard, que le navigateur utilisera en derniers recours s'il n'a pas réussi à trouver une des polices précédentes.

ASTUCE Comment obtenir tel ou tel effet ?

Si vous vous demandez comment l'on obtient l'effet par défaut sur tel ou tel élément (par exemple, quels sont les paramètres que le thème en cours utilise pour styler les boutons, les zones de texte...), il vous suffit de prendre l'archive .jar du thème (généralement disponible dans le dossier chrome de l'installation du navigateur, ou dans votre dossier personnel (par exemple : `~\Application Data\Mozilla\Profiles\default\qwgopars.slt\chrome`) et de la renommer en .zip pour pouvoir en extraire le contenu. Dans le dossier global de l'archive, vous pourrez alors examiner quels sont les effets appliqués par le thème à tel ou tel élément et éventuellement les réutiliser pour XUL Forum.

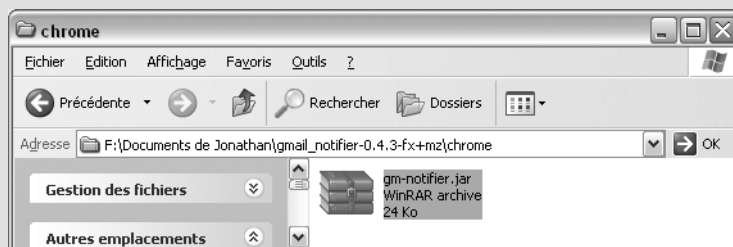


Figure 6-4 Si vous n'avez pas accès directement au fichier .jar, il faut d'abord ouvrir le fichier .xpi (au préalable renommé en .zip)

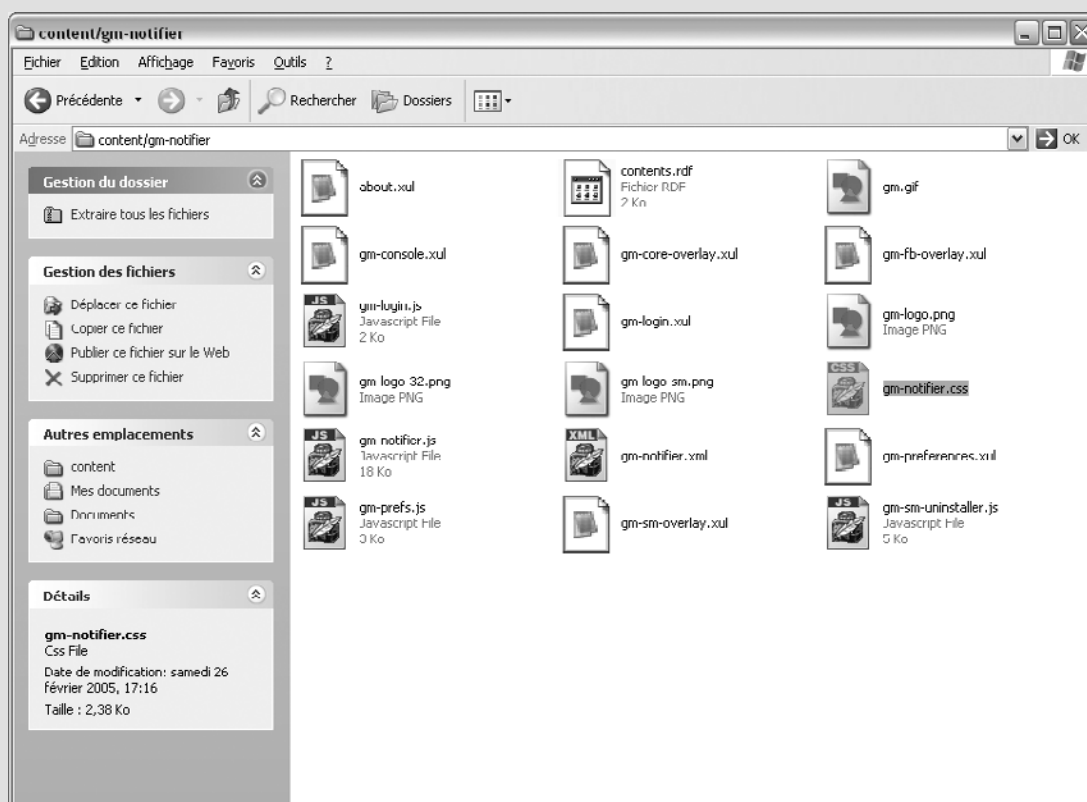


Figure 6-5 Ensuite, dans le dossier chrome, on trouve un fichier .jar qui, une fois renommé en .zip et étudié, nous fournit l'intégralité des fichiers de l'extension

Retoucher le positionnement avec CSS

Après les effets de texte, on peut encore trouver quelques points à améliorer : marges, bordures, espacement entre éléments peuvent être retouchés.

Les panneaux pour les membres sont par exemple collés au *splitter* à droite et à la limite de l'écran à gauche. On peut les décoller des bords en forçant des marges à gauche et à droite :

```
tabbox {
  margin-left: 5px;
  margin-right: 5px;
}
```

Figure 6-6

Sur cette capture d'écran, on peut voir l'espace libéré à gauche et à droite

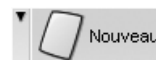


On peut aussi rajouter un texte associé aux boutons de la barre d'outils et utiliser CSS pour forcer le texte à être placé sous le bouton.

```
<toolbarbutton id="index-toolbar-nouveau"
  tooltiptext="&index.toolbar_nouveau;"
  label="Nouveau" />
```

Avec ce code seul, le texte *Nouveau* sera placé juste à droite du bouton.

Figure 6-7



Il manque la définition CSS suivante :

```
toolbarbutton {
  -moz-box-orient: vertical;
}
```

L'attribut `-moz-box-orient` force le bouton de la barre d'outils à être un conteneur vertical. Car l'élément `toolbarbutton` est en fait un conteneur, qui contient une image cliquable et un texte.

Mais `-moz-box-orient` n'est pas défini dans les standards du CSS : en fait c'est une extension Mozilla à CSS. Nous allons reprendre l'exemple du bouton de la barre d'outils pour l'expliquer en détail.

CSS spécifique à Mozilla

La barre d'outils

Pour la barre d'outils, nous allons personnaliser les boutons de telle sorte qu'au passage de la souris, leur aspect change : ils seront mis en surbrillance.

Si vous regardez le fichier CSS `chrome://navigator/skin/navigator.css` (pour la suite Mozilla), vous pourrez étudier les différentes propriétés appliquées aux boutons de la barre d'outils. En voici un exemple (issues du thème Orbit) :

```
#back-button {
  list-style-image: url("chrome://navigator/skin/o3-imag-nav-back.png");
  -moz-image-region: rect(0px 46px 42px 0px);
}
```

Pour l'élément d'id `back-button` (le bouton précédent de la barre d'outils), le thème précise l'URL de l'image à utiliser pour ce bouton. À partir de maintenant, nous allons préciser les images à afficher pour les boutons de la barre d'outils directement dans le fichier CSS. Ceci évite, lors d'un changement d'aspect de l'application, de devoir toucher aux fichiers XUL. Ainsi, si on change totalement le thème de XUL Forum, images comprises, nous n'aurons que les fichiers CSS à modifier (et de plus, les images seront placées dans le dossier skin, ce qui simplifie le tout).

La propriété `list-style-image` indique l'image à utiliser pour la puce dans un élément de type liste. L'image remplace le rond typiquement utilisé en HTML (mais par défaut invisible en XUL). Ici, l'élément `toolbarbutton` est un élément de type liste et son texte est spécifié via l'attribut `label` du fichier XUL. Son image est quant à elle spécifiée dans le fichier CSS. Si l'on précise le texte et l'image, sans attributs supplémentaires, on aura un effet similaire à celui visible dans l'illustration.

Cet effet sera comme nous l'avons vu annulé avec la directive `-moz-box-orient: vertical;`.

Si vous avez la curiosité d'aller voir comment est constituée l'image du bouton, vous remarquerez qu'en fait elle contient quatre images collées les une aux autres : ce sont les différents états du bouton.



Figure 6-8



Figure 6-9

Les quatre états du bouton précédent : normal, survolé par la souris, enfoncé et désactivé

Pour un vrai thème Mozilla, il y en a quatre : état normal, état survolé par la souris, état pressé (lorsque le bouton de la souris est enfoncé) et état désactivé. Pour XUL Forum, nous n'en garderons que deux.

Enfin, la directive `-moz-image-region` indique que la portion d'image utilisée sera un rectangle, exprimé sous la forme `rect(a, b, c, d)`.

- `a` est la distance entre le haut de l'image et le haut de la portion à retenir ;
- `b` est la distance entre le bord gauche de l'image et le bord droit de la portion à retenir ;
- `c` est la distance entre le haut de l'image et le bas de la portion à retenir ;
- `d` est la distance entre le bord gauche de l'image et le côté gauche de la portion à retenir.

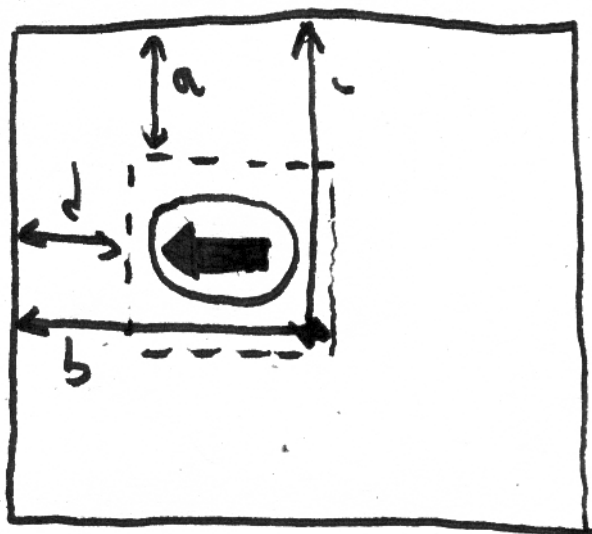


Figure 6-10
Les grandeurs `a`, `b`, `c`, `d` utilisées
pour définir un rectangle.



Figure 6-11
Le type d'icône utilisé pour XUL Forum

Ainsi, nous pouvons récapituler tous ces attributs dans notre style général qui sera appliqué à tous les éléments `toolbarbutton` :

```
toolbarbutton {
  -moz-image-region: rect(0px 32px 32px 0px);
  -moz-box-orient: vertical;
  border: 0 !important;
  font-size: xx-small;
}
```

On précise que l'on ne veut sélectionner que le rectangle de 32×32 supérieur de l'image, que l'image et le texte du bouton doivent être superposés et qu'il n'y a pas de bordure. L'attribut de taille de police concerne le texte de légende placé sous l'image du bouton. En choisissant une petite taille, on le rend plus discret, moins flagrant que les menus, car il a en effet une importance moindre.

Comme toutes nos images seront constituées de la même façon (les deux images de 32 pixels sur 32 superposées), il est possible de spécifier la portion d'image à utiliser directement dans le sélecteur d'éléments de type `toolbarbutton`, ce que nous faisons ici.

Ainsi, pour les `toolbarbuttons` survolés par la souris, nous afficherons la deuxième image, celle placée en dessous. Nous soulignerons aussi la légende du bouton lorsque l'on passe la souris sur le bouton.

```
toolbarbutton:hover {
  -moz-image-region: rect(32px 32px 64px 0px);
  text-decoration: underline;
}
```

Il reste à définir pour chaque bouton l'image qui lui correspond.

```
toolbarbutton#index-toolbar-nouveau {
  list-style-image:
    url("chrome://xulforum/skin/icones/new.png.double");
}

toolbarbutton#index-toolbar-editer {
  list-style-image:
    url("chrome://xulforum/skin/icones/edit.png.double");
}

/* ... propriétés pour les autres boutons ... */
```

Autres propriétés CSS

Les onglets du panneau de gauche

Nous allons, grâce à CSS, rajouter un effet sur les onglets, qui ne s'adressera qu'à l'onglet actif. En effet, Mozilla met à jour dynamiquement la propriété `selected` des éléments `tab`, à chaque changement d'onglet actif. Si sa valeur est `true`, l'onglet sera sélectionné. Dans le cas inverse, la valeur de cet attribut sera `false`. Vous pouvez d'ailleurs spécifier ces attributs dans le fichier XUL, pour sélectionner dès le départ un onglet autre que le premier de la liste ; vous pouvez ainsi mettre sur le deuxième onglet `selected="true"`.

ASTUCE L'attribut !important

Lorsque, parmi les multiples spécifications (feuille de style basique, thème, feuille de style de l'utilisateur, feuille de style de XUL Forum), on veut qu'un élément ait plus d'importance que d'autres, même si ceux-ci sont placés plus haut dans la cascade des CSS, on lui accole le mot-clé `!important`. Ainsi, lorsque nous voulons enlever la bordure, nous écrasons à coup sûr tout ce qui a été spécifié auparavant. Cette astuce peut être combinée avec une autre, pour éviter que le thème n'impose ses choix : la propriété `-moz-appearance: none` ; qui annule les éventuelles améliorations destinées à mieux intégrer XUL dans le thème du système d'exploitation. Les deux peuvent être combinés comme ceci :

```
-moz-appearance: none !important;
```

Nous allons donc spécifier dans le fichier CSS le style qui s'appliquera aux éléments `tab` portant l'attribut `selected="true"`.

```
tab[selected="true"] {
    text-decoration: underline;
    color: #036;
}
```

Ce nouveau sélecteur qui fait son apparition permet de sélectionner, parmi les éléments `tab`, ceux dont l'attribut `selected` a la valeur désirée. On les souligne et on choisit une couleur similaire à celle des titres.

Couleur des lignes de l'arbre

Pour pousser la personnalisation de l'interface un peu plus loin, nous allons spécifier des couleurs alternées pour les différentes lignes de l'arbre. Une ligne sur deux sera légèrement grisée pour simplifier la visualisation des autres informations comme la date, le nombre de réponses, etc.

Le CSS sera un peu différent de ce que nous avons vu précédemment. En fait, tous les éléments `treecell`, `treerow` sont purement structurels. Concrètement, ce ne sont que les éléments `treechildren` qui ont une représentation graphique. Nous allons donc utiliser un sélecteur spécial pour les éléments `treechildren`.

```
treechildren::-moz-tree-row(odd) { background-color: #EEEEEE; }
treechildren::-moz-tree-row(selected) { background-color:
    #ffe3ff; }
treechildren::-moz-tree-cell-text(selected) { color:
    rgb(26%, 31%, 39%); }
```

RAPPEL Pseudo-classe

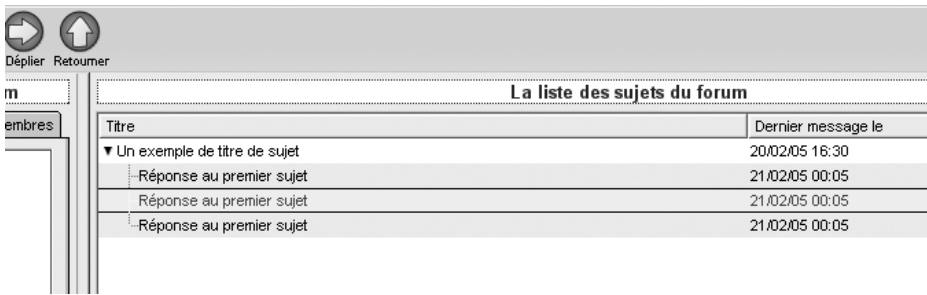
Pour une liste complète et des explications plus détaillées sur ce concept CSS, l'annexe D est à votre disposition.

La pseudo-classe `:-moz-tree-row` sélectionne les `treechildren` constituant une ligne. Le paramètre `odd`, signifiant « impair » en anglais, indique qu'on choisit les lignes impaires. On leur applique une couleur de fond gris clair.

Quand une ligne est sélectionnée, sa couleur de fond devient rose. Il est possible de combiner différents paramètres pour, par exemple, choisir une couleur de fond pour les lignes sélectionnées impaires et une couleur de fond pour les lignes sélectionnées paires. On utilise alors :

```
treechildren::-moz-tree-row(selected, odd) {}
```

Enfin, pour les textes des cellules, lorsque ces dernières sont en état sélectionné, on choisit une couleur de texte bleu marine.



Titre	Dernier message le
▼ Un exemple de titre de sujet	20/02/05 16:30
Réponse au premier sujet	21/02/05 00:05
Réponse au premier sujet	21/02/05 00:05
Réponse au premier sujet	21/02/05 00:05

← Ligne sélectionnée

Figure 6-12 Le rendu final pour l'arbre :
la ligne sélectionnée est en rose et les lignes impaires sont en gris

On peut également créer soi-même ses propriétés pour un élément de l'arbre. Ainsi, on peut définir une cellule de cette manière :

```
<treecell properties="nonlu" label="Réponse au premier sujet" />
```

Il est alors possible de choisir une image pour les cellules de type « non lu », dans le fichier CSS à l'aide de la propriété `list-style-image`.

```
treechildren::-moz-tree-image(nonlu) {
    list-style-image:
        url("chrome://xulforum/content/images/non-lu.png");
    margin-right: 2px;
}
```

Ceci sera particulièrement utile lors de la création de l'arbre : il suffira juste de mettre la bonne valeur à l'attribut `properties` pour avoir immédiatement l'icône correspondante. Et par la suite, avec un peu de JavaScript, dès qu'un message sera lu, nous pourrons lui enlever sa propriété `nonlu` et lui rendre de nouveau son aspect normal.

Enfin, pour la liste des membres, on peut réutiliser les couleurs de l'arbre, avec le CSS suivant, qui mettra les lignes sélectionnées en rose :

```
listitem[selected="true"] {
    color: rgb(26%, 31%, 39%);
    background: #ffe3ff;
}
```

Images dans les menus

Nous avons vu rapidement dans un chapitre précédent qu'il était possible d'associer une icône à un élément de menu. Nous avons alors utilisé la propriété `image` et spécifié l'URL de l'image directement dans le fichier XUL. Maintenant que la technologie CSS est exploitée dans

notre application, nous pouvons utiliser la désormais célèbre propriété `list-style-image` pour choisir l'URL de l'image depuis le skin.

```
<menuitem label="&index.menu_fichier_nouveau;"
  class="menuitem-iconic"
  src="chrome://xulforum/content/images/menu-new.png"
  id="xf-menu-fichier-nouveau"/>
```

Voilà comment nous avons obtenu la petite icône placée dans l'entrée de menu *Fichier>Nouveau*, grâce à une technique peu élégante. Nous pouvons maintenant supprimer l'attribut `src` et ajouter dans `xulforum.css` :

```
#xf-menu-fichier-nouveau {
  list-style-image:
    url("chrome://xulforum/content/images/menu-new.png");
}
```

Utilisation d'une propriété propre à Mozilla

Enfin, pour illustrer les possibilités offertes par les extensions CSS de Mozilla, on peut courber légèrement le cadre de l'écran d'authentification :

```
groupbox {
  -moz-border-radius: 5%;
}
```

Selon votre thème, vous aurez soit des bords arrondis au cadre, soit des angles brisés. C'est en quelque sorte la touche finale à notre parcours du CSS.

En résumé...

Nous avons terminé l'interface de XUL Forum. Maintenant que la couche de CSS est en place, à la fois la structure et l'apparence sont finies.

- Des mises en formes en apparence simples ont permis de mettre en relief différents titres dans les deux pages actuelles de XUL Forum.
- Nous avons ensuite retouché de manière plus structurée les marges ainsi que l'espacement entre les différents éléments.
- Des mises en forme propres à Mozilla ont modifié la barre d'outils, pour rajouter du texte sous les boutons ou pour changer l'aspect d'un bouton au passage de la souris.
- Enfin, les mises en forme finales ont rajouté des icônes dans les menus, des couleurs dans les listes et dans l'arbre.

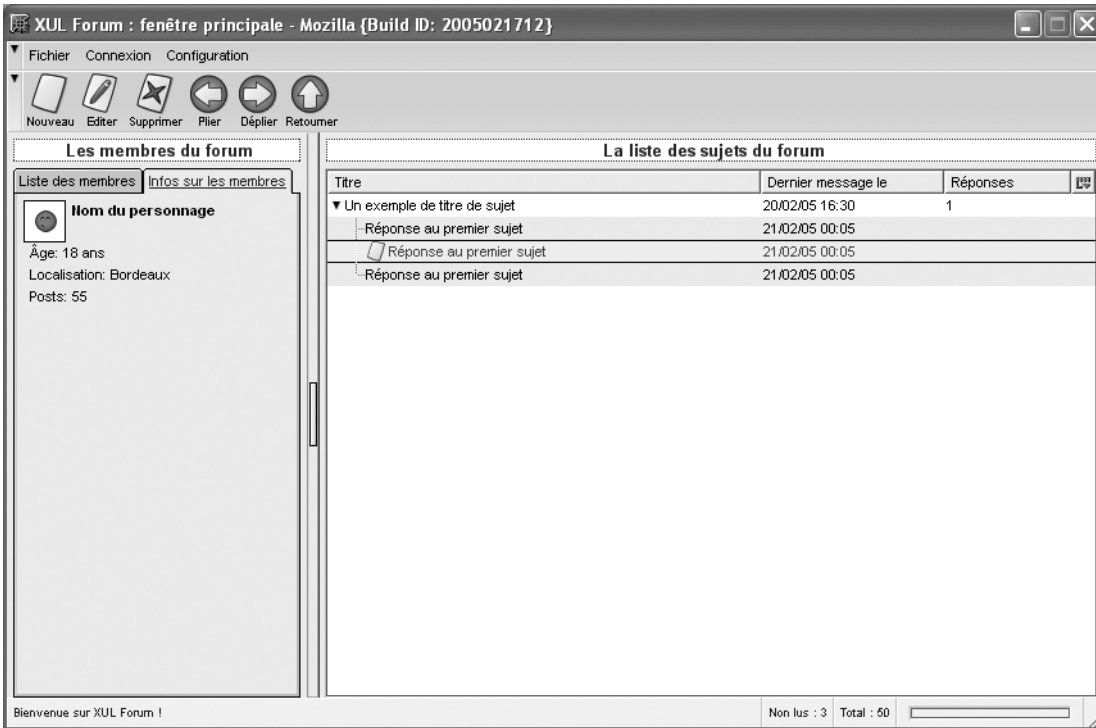
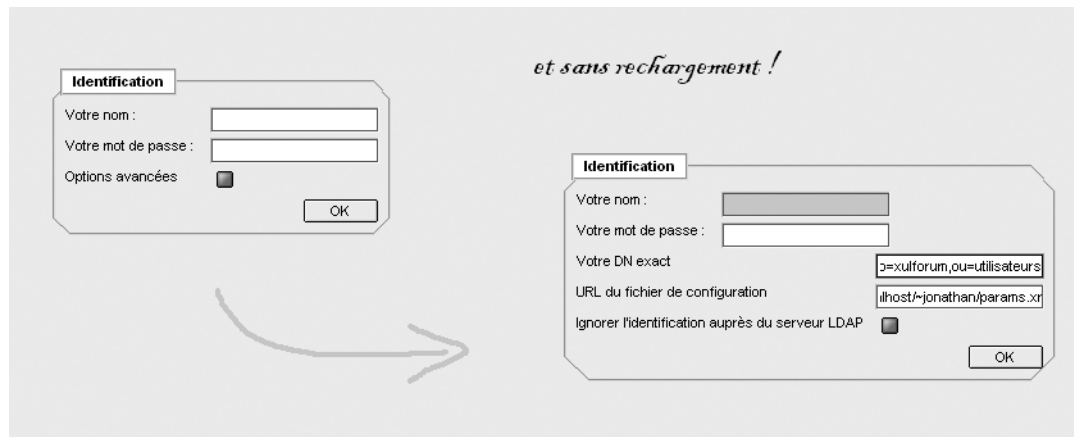


Figure 6–13

La partie statique est désormais terminée : il est grand temps de rendre cette application un peu plus dynamique grâce à JavaScript !

chapitre 7



Première animation de l'interface avec JavaScript

Notre application est maintenant fort agréable à l'œil, mais terriblement figée. En effet, il n'y a rien de dynamique ! Nous allons remédier à cela grâce à JavaScript...

SOMMAIRE

- ▶ Concepts de base du langage
- ▶ Une première manipulation d'éléments
- ▶ Récupération d'un fichier de configuration extérieur
- ▶ Première utilisation d'un composant XPCOM

MOTS-CLÉS

- ▶ DOM
- ▶ Instanciation de composants XPCOM
- ▶ Piles

Dans les chapitres précédents, nous avons successivement mis en place l'application dans le contexte du navigateur, réalisé les différentes pages de XUL Forum, mis de l'ordre dans l'apparence avec CSS, séparé les éléments communs à toutes les pages, mais nous n'avons toujours qu'un XUL Forum figé, incapable d'interagir avec son environnement.

C'est ici que JavaScript intervient : le langage bien connu des développeurs web est bien plus puissant qu'il n'y paraît et nous allons nous attacher à faire bien plus qu'un inopportun *pop-up* publicitaire !

Concepts de base du langage

Appelé LiveScript durant sa phase de développement, JavaScript connut sa première version publiée, la 1.0, avec la sortie de Netscape Navigator 2.0. Ce langage a ensuite été implémenté au sein du navigateur Internet Explorer de Microsoft sous le nom de JScript, avec quelques variantes. Pour répondre au problème d'implémentations concurrentes, avec chacune leurs spécificités, un organisme de standardisation, l'ECMA, a publié différentes versions de la norme ECMA-262, qui standardise le langage sous le nom d'ECMAScript. La standardisation comprend des éléments comme les structures `switch`, les exceptions, etc.

À l'heure actuelle, les navigateurs de la fondation Mozilla utilisent SpiderMonkey, l'implémentation JavaScript de Mozilla écrite en langage C. Dérivée du projet initial de Netscape, elle porte le numéro de version 1.5 et est conforme à la norme ECMA-262 v3. Les deux autres implémentations majeures, celle d'Opera et de Microsoft, JScript 1.5, sont elles aussi conformes à la norme.

Parallèlement à JavaScript existe le DOM. Il ne faut pas confondre les deux. Le DOM est une recommandation du W3C qui définit de façon neutre la manière d'accéder au contenu d'un document HTML ou XML (ce qui concerne aussi les XUL, naturellement), de le manipuler, de le modifier et de gérer des événements. Contrairement à JavaScript, la maturité des implémentations du DOM varie selon les navigateurs, la plus conforme à l'heure actuelle étant celle de Mozilla. Il existe des API DOM pour tous les langages : C, PHP, etc.

Une des idées reçues les plus tenaces consiste à affirmer que JavaScript et Java sont similaires. Il n'en est rien ! Même s'il existe des ressemblances syntaxiques, le nom de JavaScript a été choisi lors d'un communiqué de presse commun à Netscape et à Sun Microsystems, pour des raisons commerciales diront certains. Quoi qu'il en soit, les deux langages sont indépendants. Ils peuvent cependant se compléter et JavaScript peut contrôler une applet Java.

► <http://www.ecma-international.org>

ATTENTION DOM avec PHP4 et PHP5

Nous serons amenés à utiliser PHP5 plus en avant dans ce livre. Il faut faire attention à l'interprétation que l'on fait du terme « extension DOM » en parlant de PHP. Le DOM est normatif : les langages qui respectent cette recommandation garantissent des fonctions, des attributs, des méthodes identiques. Ainsi, la manipulation DOM se fera de la même façon entre JavaScript et PHP5, à la syntaxe du langage près : PHP5 et JavaScript sont tous deux des langages qui respectent la norme DOM. En revanche, PHP4 n'est pas conforme à la norme DOM. Il est donc déconseillé d'utiliser ce langage pour manipuler des documents XML.

Beaucoup de gens pensent aussi que JavaScript est un langage simple, qui ne sert qu'à ouvrir des fenêtres publicitaires dans les pages web. Nous allons nous efforcer de démontrer au travers de cet ouvrage que JavaScript soutient aisément la comparaison avec d'autres langages : couche objet, exceptions, sécurité, en font un candidat de choix pour le programmeur.

Syntaxe de base

Nous allons parcourir rapidement les caractéristiques de ce langage, afin de nous remettre en mémoire sa syntaxe. Le but ici n'est pas d'être exhaustif, au contraire, mais de mettre rapidement le lecteur en situation, afin de lui permettre de saisir l'utilité de JavaScript dans le cadre de XUL.

Pour vous former en JavaScript, ou obtenir une référence, il existe des ouvrages spécifiques. Cependant, avec des connaissances de base en programmation, la transition ne devrait pas être trop difficile !

 JavaScript, la référence chez O'Reilly

Variables, utilisation de fonctions

Les instructions en JavaScript sont toutes terminées par un point-virgule, comme en C ou en Java.

```
dump("JavaScript rox0rz");
var err = "Erreur !";
```

Les variables sont déclarées grâce au mot-clé `var`. Le langage n'est pas typé, c'est-à-dire que lorsque vous créez une variable, vous pouvez lui faire contenir du texte, un nombre, un objet, toujours en écrivant `variable = quelquechose`;

Les fonctions comme `dump()` dans l'exemple ci-dessus s'utilisent de manière courante, c'est-à-dire `fonction(argument1, argument2, etc.)`;

Commentaires

Les commentaires, de même qu'en C++, s'écrivent de la manière suivante :

```
//un commentaire qui s'étale sur une ligne seulement
/* commentaire qui peut
   prendre
   plusieurs
   lignes
*/
```

Chaînes

Les chaînes sont placées entre guillemets doubles ou simples et se concatènent grâce à l'opérateur +.

```
var machaine = "J'additionne" + "plusieurs" + "chaînes";
var autrechaîne = "J'additionne" +
  "sur" + "plusieurs lignes";
var erreur = "Je ne peux pas
  couper ma chaîne sur deux lignes
  sans guillemets";
```

La dernière ligne provoque quant à elle une erreur car il est indispensable de fermer ses chaînes avant le saut de ligne (comme en C).

Déclarations de fonctions

Les fonctions se déclarent comme en C, sauf que l'on omet les types, le langage étant non-typé.

```
function foo(bar) {
  dump(bar);
}
```

`dump` est une fonction qui agit sur des chaînes : l'argument `bar` de la fonction `foo` sera automatiquement converti en chaîne au moment de le passer à `dump()`.

OUTILS Débogage efficace en JavaScript

Il est indispensable d'avoir les bons outils pour programmer efficacement en JavaScript. Le premier d'entre eux est la console JavaScript, qui affiche toutes les erreurs non gérées par le programmeur. Elle se lance de deux manières : au démarrage, via l'option de la ligne de commande `-jsconsole`, ou dans Mozilla, dans les menus *Outils/Options/Développement Web*.

En dehors de la console JavaScript, il existe aussi la sortie standard ; c'est la fonction JavaScript `dump()` qui permet d'y afficher du texte. Cette fonction n'est disponible que si vous avez suivi les instructions du chapitre 5, pour activer les fonctionnalités de débogage, par l'intermédiaire de votre fichier `prefs.js`. Sous Unix, la sortie standard est accessible en lançant Mozilla depuis un shell. Sous Microsoft Windows, vous devrez demander à Mozilla d'ouvrir une fenêtre MS-Dos grâce à l'option de la ligne de commande `-console`.

Cette technique se révèle très utile, car elle permet d'afficher des informations durant la phase de développement uniquement, l'utilisateur final n'ayant généralement ni activé les fonctionnalités de débogage, ni accès à la sortie standard.

Enfin, le débogueur JavaScript Venkman, inclus dans Mozilla et disponible sous forme d'extension pour Firefox, vous permet de manipuler dynamiquement votre document, pour essayer différentes propriétés du DOM, sans avoir à recharger continuellement la page. C'est un interpréteur direct de code JavaScript. Il existe aussi des fonction de débogage à proprement parler, avec des *breakpoints*, mais ceux-ci ne semblent pas fonctionner avec des fichiers XUL (uniquement avec des fichiers HTML).

► http://www.svendtofte.com/code/learning_venkman/

Méthodes d'objets

En ce qui concerne les objets, ils sont partout ! On accède à la méthode d'un objet grâce à l'opérateur point : `objet.maméthode()`. Ainsi, pour arrondir un nombre, on utilisera la méthode `toFixed` de l'objet `Number` (tous les nombres sont des instances de l'objet `Number`, toutes les chaînes sont des instances de l'objet `String`, etc.). Ceci est le même principe qu'en Java : tous les types primaires sont convertis en objets le moment venu (technique d'autoboxing).

```
var n = 3.14159;
var pi = n.toFixed(3); //pi est une chaîne
dump("Pi vaut : "+pi+" à trois chiffres significatifs\n");
/* insertion ci-dessus d'une nouvelle ligne grâce au caractère
spécial \n, à insérer entre doubles guillemets */
```

ATTENTION Langage non typé

Il ne faut pas se méprendre : les variables ont bien un type, qui peut être soit un nombre, soit une chaîne. On peut d'ailleurs l'obtenir grâce à la fonction `typeof()`. Mais le programmeur n'a généralement pas à s'en soucier, car les conversions sont dans la plupart des cas automatiques. Ainsi, dans l'exemple ci-contre, `n`, qui est un nombre, serait converti automatiquement en chaîne si on le mettait à la place de `pi` : `"..." + n` transforme `n` en instance de l'objet `String`. Si l'on veut être plus précis, on dira que c'est grâce à la méthode `toString` de la classe `Number`.

Tableaux

Les tableaux sont des instances de la classe `Array`. Il n'existe pas de tableaux associatifs (on utilisera plutôt des objets) ; les tableaux disposent donc uniquement d'index.

```
var a = new Array();
a[0] = "un élément";
a[1] = 3;

var a = new Array(5);
//tableau de 5 éléments, donc index allant de 0 à 4

var a = new Array("un élément", 1.41, 3);
```

Objets : instantiation

C'est dans son rapport au modèle objet que JavaScript diffère d'un langage traditionnel. Un objet est créé en appelant une fonction et cette fonction a le droit d'accéder à l'objet `this`, pour éventuellement le remplir (propriétés, méthodes). En voici un exemple :

```
function func1() {
    dump("Méthode !");
}

function foo(bar) {
    this.bar = bar;
    this.func1 = func1;
}

var mabarre = new foo("bar");
```

La fonction `foo` est utilisée ici comme le constructeur de l'objet `foo` et peut donc accéder à `this`. Elle définit une propriété pour l'objet et une méthode, qui est en fait une variable contenant une référence vers une autre fonction. On pourra ainsi utiliser `dump(foo.bar)` et `foo.func1()`.

Nous n'utiliserons que rarement cette syntaxe pour déclarer nos objets. Il en existe deux autres, que nous verrons plus loin.

Exceptions

JavaScript est bel et bien un langage évolué : il a sa propre gestion des exceptions.

```
try {
    throw new Exception(
        "l'exception n'est pas toujours celle que l'on croit");
} catch (e) {
    alert("Erreur ! "+e);
} finally {
    //instructions exécutées quoi qu'il arrive
}
```

Ici s'affichera un message d'erreur dans une boîte de dialogue : « Reference error, class Exception not defined ». En effet, la classe `Exception` n'existe pas et le moteur de JavaScript nous lance une erreur, car nous voulons utiliser une classe qui n'existe pas ! Et l'exception n'est pas celle que nous avons demandée, c'est une autre qui proteste contre l'absence de classe `Exception`.

La classe correcte est la classe `Error`.

```
try {
    throw new Error("Erreur de notre part");
} catch (e) {
    alert("Erreur ! "+e);
} finally {
    //instructions exécutées quoi qu'il arrive
}
```

Ici, nous aurons bien notre message d'erreur qui s'affichera !

Plus...

Il existe bien sûr les tests conditionnels, les boucles `switch`, les syntaxes alternatives pour les objets, etc. Nous verrons tous ces points particuliers au fur et à mesure de l'application, ce qui précède ne servant qu'à jeter les bases afin de pouvoir ensuite continuer à apprendre tout en commençant à programmer.

Intégration à XUL

On peut directement intégrer un script dans un document XUL de deux manières :

```
...
<window>
  <script type="application/x-javascript">
    <![CDATA[
      ...
    ]]>
  </script>
</window>
...
```

Dans ce cas de figure, le contenu du script sera directement lié au document XUL, ce qui est en général une pratique à éviter.

L'attribut `type` indique le contenu de la balise `script` qui va suivre. Il est possible de l'omettre. Le contenu du script est encadré entre les chaînes `<![CDATA[` et `]]>`. Ces instructions spéciales sont à destination du parseur XML et lui indiquent que le contenu qu'elles encadrent n'est pas du XML et, par conséquent, qu'il ne doit pas être analysé : le parseur ne doit pas y chercher des balises, simplement les représenter en tant que données (« character data ») dans son arbre du document. Ceci évite des erreurs d'analyse pour les signes de comparaison comme `>` ou `<`, qui peuvent être interprétés comme annonçant des balises XML, ou pour les signes comme `&`, qui annoncent une entité.

Une autre manière, plus élégante, consiste à simplement inclure un script depuis un fichier XUL, en utilisant l'attribut `src` de la balise `script`.

```
...
<window>
  <script
    src="chrome://xulforum/content/javascript/identification.js"
    type="application/x-javascript" />
</window>
```

On retrouve les avantages habituels à cette solution : modularité, utilisation d'URL `chrome://`, lisibilité, etc. Nous incluons donc cette ligne dans notre fichier `xulforum.xul` et nous allons commencer nos premières lignes de JavaScript dans XUL Forum !

Dans ces fichiers JavaScript, nous retrouverons surtout des routines : gestion d'erreurs, manipulation de l'interface et gestionnaires d'événements, capables de réagir aux actions de l'utilisateur. Les lecteurs familiers du HTML pourront ainsi utiliser des attributs comme `onclick="maFonctionJS()"` sur des éléments XUL et créer la fonction correspondante dans le fichier JavaScript.

Application directe à XUL Forum

Une première routine pour l'affichage d'erreurs

ATTENTION Clarté des messages d'erreur

Une pratique fort désagréable consiste à signaler à l'utilisateur toute erreur par un appel à la fonction `alert()`. Outre le fait que ceci devient rapidement agaçant, l'utilisateur ne prête plus attention au contenu du message et se contente de cliquer sur le bouton *OK*. N'abusez donc pas de cette solution ! Nous allons au contraire utiliser la barre de statut pour afficher nos messages, discrètement.

Nous allons tout d'abord programmer des routines pour gérer les messages d'erreur : une fonction `ajouterErreur()` pour ajouter un nouveau message d'erreur et une fonction `afficherErreur()` appelée à intervalles réguliers pour mettre à jour la barre de statut.

Après avoir ajouté la balise `<script>` à `xulforum.xul`, nous pouvons éditer le fichier `content/javascript/identification.js`. Il n'y a pas besoin de formatage spécial du fichier, nous pouvons ainsi directement écrire :

```
dump("JavaScript lancé !\n");
//ou plus voyant
alert("Démarrage du JavaScript");
```

La première solution affichera le message sur la console tandis que la seconde ouvrira une boîte de dialogue, contenant un point d'exclamation.

Voici le code concernant notre système de gestion d'erreurs. L'idée est de gérer un tableau global, une pile, contenant les messages d'erreurs. Chaque nouveau message d'erreur est ajouté en haut de la pile et la fonction `afficherErreur()`, appelée à intervalles réguliers, prend l'élément le plus en bas de la pile pour l'afficher. Ensuite (deux secondes plus tard par exemple), cette même fonction prend le nouvel élément le plus en bas de la pile et l'affiche à son tour. Ceci permet d'avoir un délai minimum entre deux messages d'erreur, pour laisser à l'utilisateur le temps de les lire.

ASTUCE Où trouver de l'aide sur le JavaScript dans Mozilla ?

Autrefois, la référence était DevEdge, le site de Netscape. On y trouvait des *sidebars* pour Mozilla, la référence JavaScript 1.5... Mais Netscape a fermé ce portail. Après des tractations entre Mozilla et Netscape, la fondation Mozilla a récupéré l'intégralité du contenu.

On retrouve donc d'une part l'ancien miroir de DevEdge (amené, à terme, à disparaître au profit du site `developer.mozilla.org`). On peut chercher sur ce site avec le moteur Google, en tapant parmi vos mots-clés de recherche la chaîne « `site:devedge-temp.mozilla.org` ».

► http://devedge-temp.mozilla.org/index_en.html

Il y a d'autre part le serveur wiki de `devmo` (« `developer mozilla` »), où se concentre le nouvel effort de documentation. On y retrouve notamment la référence JavaScript 1.5.

► http://developer-test.mozilla.org/en/docs/Main_Page
 ► http://developer-test.mozilla.org/en/docs/Core_JavaScript_1.5_Reference

Enfin, pour les personnes désireuses de suivre l'actualité du projet, c'est-à-dire les propositions, les évolutions et tout ce qui concerne les discussions entre développeurs (pour Mozilla 2.0 par exemple), il y a le wiki de développement.

► http://wiki.mozilla.org/Main_Page

Outils Enregistrer les fichiers .js dans le bon encodage

Les fichiers JavaScript ne sont pas des fichiers XML : Mozilla s'attend à avoir affaire à des fichiers encodés en latin1. Pensez à désactiver la sauvegarde en Unicode, sinon vous risquez d'avoir des symboles étranges dans votre barre de statut à la place des accents. Commande utile pour (g)vim :

```
:set fileencoding=latin1
:set encoding=latin1
```

```
/* tout ce qui concerne la gestion des erreurs sur la barre de statut */
var gErreurs = new Array; ❶
const DELAI = 2000; ❷

function ajouterErreur(pErreur) {
    var d = new Date(); ❸
    gErreurs.push({erreur: pErreur, date: d}); ❹
}

function afficherErreur() {
    var e;
    if ((e = gErreurs.shift()) != null) { ❺
        document.getElementById("xf-statusbar-status").label =
            e.date.toLocaleTimeString()+" : "+ e.erreur; ❻
    } else {
        document.getElementById("xf-statusbar-status").label = "Ok"; ❼
    }
    setTimeout(afficherErreur, DELAI); ❽
}

setTimeout(afficherErreur, DELAI); ❾
ajouterErreur("JavaScript démarré"); ❿
```

Le fonctionnement de cet exemple est un peu complexe : il mérite quelques explications.

- ❶ On déclare d'abord le tableau global, qui servira de pile et qui contiendra les différents messages d'erreur.
- ❷ La constante DELAI contient l'intervalle de temps, en millisecondes, qui s'écoulera entre deux mises à jour de la barre de statut.
- ❸ La fonction ajouterErreur est appelée à chaque nouvelle erreur (ou message de statut) à afficher. La première variable est utilisée pour stocker l'heure à laquelle l'erreur se produit : c'est un objet Date.
- ❹ On ajoute un nouvel élément au sommet du tableau, grâce à la méthode push(). Cet élément que l'on ajoute est un objet, déclaré grâce à la syntaxe entre accolades. Elle suit la règle suivante : {variable1: valeur1, variable2 : valeur2...}. Ici, on ajoute au sommet de la pile d'erreurs un objet ; sa variable membre appelée

erreur représente le message d'erreur et une autre variable membre `date` est un objet `Date` créé au moment où l'erreur s'est produite.

- ⑤ La fonction `afficherErreur` sera appelée toutes les deux secondes (c'est le délai que nous avons choisi). On commence par voir si l'on a un élément non nul au bas de la pile. La syntaxe est ici raccourcie : on aurait pu écrire :

```
var e = gErreurs.shift(); if (e != null) {...}
```

- ⑥ Si effectivement on a un message d'erreur en attente d'être affiché, on récupère l'élément portant l'ID `xf-statusbar-status` (l'élément XUL `<statusbarpanel id="xf-statusbar-status" label="Bienvenue sur XUL Forum !" flex="1" />`) et on change sa propriété `label`. Le résultat est « 17:01:52 : JavaScript démarré », qui est affiché en bas de l'écran.
- ⑦ Dans le cas contraire, on affiche un message par défaut.
- ⑧ On met un minuteur qui va de nouveau appeler notre fonction dans deux secondes.
- ⑨ On amorce la fonction : il faut qu'elle soit appelée une fois pour pouvoir ensuite se rappeler elle-même.
- ⑩ On affiche un premier message, indiquant que le JavaScript a commencé son travail.

Ainsi, nous pouvons afficher des messages d'erreur sur la barre de statut, tout en sachant qu'ils seront au moins séparés par deux secondes, le temps pour l'utilisateur de les lire.

Mais un nouveau problème se pose, celui de la localisation. Dans nos fichiers XUL, nous avons utilisé les DTD pour gérer les différentes langues, mais dans JavaScript, ce concept n'est pas applicable. Il existe un équivalent : les objets `stringbundle`.

CULTURE Piles en programmation

En programmation, le type de pile que nous avons utilisé est appelé pile FIFO : « first in, first out ». Le premier élément arrivé sortira en premier. Ce type de pile est fréquemment utilisé dans les communications avec un serveur distant lorsqu'il y a besoin de créer une file d'attente pour la transmission des messages. Nous retrouverons ce type de pile au chapitre 10 avec le proxy XPCOM pour les listeners LDAP.

Il existe un type de pile opposé : les piles LIFO (« last in first out ») ou « stack ». On empile tous les éléments sur le dessus de la pile et ensuite, on sort l'élément qui est en haut de la pile (avec par exemple en JavaScript la méthode `Array.pop()`). Ce type de pile est par exemple utilisé lors d'une analyse lexicale : d'abord on ouvre les `{`, ensuite les `[`, ensuite les `[`, puis on dépile l'élément crochet lorsqu'on rencontre son opposé `]` qui le ferme et ainsi de suite, toujours en prenant l'élément le plus en haut de la pile.

Multilangue avec l'objet `stringbundle`

La problématique est identique à celle que nous avons rencontrée avec les fichiers XUL et la solution est à peu de choses près la même.

Il faut tout d'abord créer un fichier extérieur, contenant le nom des chaînes et leur valeur, pour pouvoir exploiter le contenu de ce fichier avec JavaScript. Ce fichier sera bien sûr placé dans le dossier `locale/` `votre-locale/`, pour pouvoir y accéder via l'URL `chrome`.

Nous appellerons ce fichier `js.properties` (l'usage veut que ces fichiers portent l'extension `properties`) et nous le remplirons selon la syntaxe `nomdechaîne=contenu` de cette chaîne dans la locale choisie.

Les chaînes dans le fichier `locale/fr-FR/js.properties`

```
statutOk=Rien à signaler
statutDemarre=Script lancé
```

Pour les accents, il est indispensable de sauvegarder ce fichier en encodage UTF-8. En effet, nous allons l'inclure dans le document XUL.

Le début de `xulforum.xul` devient ainsi :

L'élément `stringbundle` qu'il faut placer dans `xulforum.xul`

```
<window ...>

  <stringbundle
    src="chrome://xulforum/locale/js.properties" id="chaines" />
  <script type="text/javascript"
    src="chrome://xulforum/content/javascript/identification.js" />
```

Il est important de placer l'élément `stringbundle` avant l'élément `script`. Ainsi, Mozilla chargera d'abord les chaînes et ensuite, le script pourra y avoir accès. Enfin, nous allons pouvoir récupérer les chaînes dans le fichier JavaScript.

La réutilisation dans le fichier JavaScript `content/javascript/identification.js`

```
var gStringBundle = document.getElementById("chaines");
...
if (...) {} else {
  document.getElementById("xf-statusbar-status").label =
    gStringBundle.getString("statutOk");
  ...
}
```

Outils Référence DOM et JS

Il existe un site référençant les méthodes et propriétés des objets du DOM, ainsi que quelques objets JavaScript. Peut-être sera-t-il plus clair pour vous que le site xulplanet.com !

► [Http://www.mozref.com](http://www.mozref.com)

C'est ici que l'on réalise toute l'utilité de placer un attribut `id` sur les éléments XUL. La variable globale `document` est une instance de l'objet `Document` et représente le document HTML, ou XUL, qui appelle le script. D'autres instances de cet objet peuvent servir à manipuler des documents XML génériques par exemple.

Et ainsi, avec le DOM, on peut utiliser la méthode `getElementById` de l'objet `Document` pour avoir accès à un élément du document et ce grâce à son identifiant. Cette méthode est utilisée deux fois, la première pour déclarer une variable globale qui représente l'élément `<stringbundle>`, la seconde pour avoir accès à l'élément de la barre de statut contenant les textes.

Comme vous l'aurez deviné, la méthode de l'élément `StringBundle` permettant d'obtenir une chaîne à partir de son nom est `getString()`.

Plus de manipulation DOM : options avancées à la connexion

Nous pouvons maintenant voir quelques fonctions du DOM pour cacher ou montrer une deuxième série d'options relatives à la connexion.

Nous allons ajouter un nouveau tableau sur notre page d'identification, qui contiendra des options avancées (nous verrons leur signification au fur et à mesure de notre progression dans JavaScript).

Le tableau d'options avancées que nous ajoutons à xulforum.xul

```
<grid style="visibility: hidden;" id="xf-grid-opt">
  <columns>
    <column id="xf-ident-opt-textes" />
    <column id="xf-ident-opt-champs" />
  </columns>
  <rows>
    <row>
      <description value="Votre DN exact" />
      <textbox id="xf-ident-opt-dn" />
    </row>
    <row>
      <description value="URL du fichier de configuration" />
      <textbox id="xf-ident-opt-config"
        value="http://www.xulforum.com/config.xml" />
    </row>
    <row>
      <description value="Ignorer l'identification LDAP" />
      <checkbox id="xf-ident-opt-ignorer" />
    </row>
  </rows>
</grid>
```

Si vous ajoutez ce code dans votre fichier `xulforum.xul` à la suite du précédent élément `<grid>`, vous remarquerez que l'espace occupé par l'élément est alloué. C'est-à-dire que tout se passe comme si Gecko plaçait d'abord tous les éléments, les dessinait tous et gommait ensuite le `grid` qui nous intéresse sans pour autant diminuer la hauteur du `groupbox`. Ceci est parfaitement normal puisque l'on demande juste à Gecko de ne pas afficher notre `grid`. On ne lui demande pas de le supprimer, ni de ne pas le prendre en compte dans la répartition des éléments !

Pour remédier à ceci, nous allons utiliser une astuce : nous allons placer notre élément `grid` en bas de la page, juste avant la barre de statut, de manière à ce qu'il ne bouscule pas l'organisation du cadre central.

Ensuite, notre JavaScript changera son emplacement et l'affichera. Ainsi, il n'y aura pas de problème d'élément fantôme occupant pourtant de l'espace.

Nous ajoutons aussi un élément dans le cadre central, une case à cocher permettant de montrer les options.

Une nouvelle ligne avec une case à cocher pour afficher les options avancées

```
...
<row id="xf-row-options">
  <description value="Voir les options avancées" />
  <checkbox id="xf-ident-opt"
    oncommand="montrerOptions();" />
</row>
</rows>
</grid>

<hbox id="xf-config-hbox">
...
```

Vous remarquerez que cette ligne a un attribut `id`, de même que la boîte horizontale plus bas (celle qui contient le bouton *OK*).

Nous avons placé sur l'élément `checkbox` un attribut `oncommand`. Il permet de spécifier un gestionnaire d'événement, c'est-à-dire la fonction qui est appelée quand l'événement `command` a lieu. Cet événement n'est pas présent dans le DOM habituel d'une page web (qui utilise plutôt `onclick`, `ondblclick`, etc.) : c'est un événement spécial, qui englobe à la fois le clic (on coche la case en cliquant dessus) et les validations clavier (on peut aussi la cocher avec la barre d'espace).

Les fonctions JavaScript qui réagissent aux événements utilisateurs sont appelées des fonctions de *callback*.

Le code servant à montrer les options avancées

La fonction JavaScript `montrerOptions()` va donc devoir effectuer les tâches suivantes :

- récupérer le tableau contenant les options avancées ;
- le placer avant l'élément `hbox` contenant le bouton `OK` ;
- le rendre visible ;
- cacher la ligne contenant la case à cocher (une fois que l'on a affiché les options, ce ne sera pas la peine de les cacher de nouveau).

```
function montrerOptions() {
    dump("Affichage des options avancées\n"); ❶
    var o = document.getElementById("xf-grid-opt"); ❷
    var g = document.getElementById("xf-config-groupbox"); ❸
    o.style.visibility = "visible"; ❹
    g.insertBefore(
        document.getElementById("xf-config-hbox"); ❺
    g.childNodes[1].childNodes[1].removeChild(
        document.getElementById("xf-row-options"); ❻
    document.getElementById(
        "xf-ident-ident_nom").setAttribute("disabled", true); ❼
}
```

CULTURE DOM

Toutes les méthodes agissant sur des conteneurs sont en fait des méthodes de l'objet DOM Node. « Node » signifie « nœud » et l'on peut voir tous les éléments XUL comme des nœuds. Un élément `Description` est un nœud qui contient un ultime nœud spécial, appelé `CharacterData`, dont la propriété `data` représente son texte.

En fait, on peut voir la logique DOM comme une hiérarchie organisée comme un arbre avec des nœuds, arbre qu'il est possible de modifier dynamiquement en agissant sur ses nœuds.

► <http://www.w3.org/DOM/>

La première chose à faire, après l'affichage ❶ d'un petit message de débogage, est de récupérer les deux principaux éléments que nous manipulerons : `o` pour le tableau des options ❷ et `g` pour le groupbox global ❸.

On rend le tableau des options visible ❹ et on l'insère comme enfant du groupbox ❺. On aurait pu utiliser la méthode `appendChild` sans second argument, mais le tableau des options serait apparu en dessous du bouton `OK`, car cette méthode ajoute l'élément en dernier. Ce n'est pas très élégant, c'est pourquoi on l'insère avant la boîte horizontale contenant le bouton `OK`, via la méthode `insertBefore()`. Vous remarquerez qu'il est nécessaire de récupérer l'élément parent pour pouvoir ensuite agir sur ses éléments enfants : c'est la logique DOM.

Puis on supprime purement et simplement l'élément `row` contenant la case à cocher ❻. Ici on montre l'utilisation du tableau `childNodes`. Numéroté à partir de 0, il contient tous les éléments fils.

- Ici, `g` est le groupbox. `g.firstChild`, est égal à `g.childNodes[0]` et représente l'élément `caption`, le titre du groupbox.
- `g.childNodes[1]` représente le grid contenant les options classiques.
- `g.childNodes[1].childNodes[0]` représente l'élément `<columns>` du grid.

- `g.childNodes[1].childNodes[1]` est l'élément `<rows>`, parent de l'élément que l'on veut supprimer. Ici on appelle la méthode `removeChild()` avec l'objet à supprimer obtenu grâce à son identifiant.

Enfin, on désactive le champ nom ⑦ puisqu'il est abandonné au profit du DN exact (DN pour Distinguished Name, l'identifiant précis de l'utilisateur auprès du serveur LDAP).

Figure 7-1
Les options avancées affichées

En conclusion, ce bref aperçu du DOM nous permet de montrer quelques-unes de ses possibilités essentielles. Nous allons en voir quelques autres en récupérant un fichier de configuration situé sur un serveur distant, au format XML. Le DOM va nous permettre de manipuler ce fichier XML pour en extraire les informations nécessaires.

ATTENTION **Cache des fichiers XUL**

Avec certaines versions de Firefox, si vous montrez les options avancées, que vous cliquez sur **OK** (pour aller sur la page suivante) puis que vous revenez à l'identification, le tableau des options avancées est de nouveau visible ! Ceci est dû à une mise en cache de la page (l'utilisateur final n'aura pas désactivé le cache XUL !). Pour éviter ceci, mettez dans votre fichier XUL :

```
<window ... onload="initialisation();" ...>
```

Puis dans la fonction `initialisation()`, dans le fichier JavaScript :

```
function initialisation () {
    document.getElementById("xf-grid-opt").style.visibility =
        "hidden";
}
```


Communication avec l'extérieur : récupération d'un fichier de configuration

L'objet XMLHttpRequest

► http://openweb.eu.org/articles/objet_xmlhttprequest/

Nous allons préparer des fichiers de configuration prédéfinis, un pour chaque forum. En fait, comme notre extension est incluse dans Mozilla, elle doit pouvoir se connecter à des forums différents, sur différents serveurs. Un fichier de configuration contiendra donc les informations nécessaires pour se connecter au serveur qu'il décrit : hôte, chemin vers le fichier pour les services web, port, etc. Chaque serveur proposant un support de XUL Forum pourra donc fournir son fichier.

ALTERNATIVE **ActiveX et Internet Explorer**

L'objet XMLHttpRequest a d'abord été inventé par Microsoft, qui le propose sous forme de contrôle ActiveX pour son navigateur Internet Explorer. Il est ainsi possible d'utiliser AJAX (présenté en tant qu'alternative au début du livre) pour les deux navigateurs, Mozilla et Internet Explorer. Les inconvénients de cette méthode ont déjà été mentionnés et nous ne nous attardons pas dessus : incompatibilité entre les différents navigateurs, code JavaScript peu « propre », HTML peu adapté à la manipulation d'interfaces graphiques, etc.

► http://www.xulplanet.com/references/objref/XMLHttpRequest.html#method_open

Pour récupérer un fichier distant, nous allons utiliser l'objet XMLHttpRequest. Son utilisation a été mise en relief avec le service d'e-mails Gmail de Google, qui se distingue par son aspect très dynamique : les pages changent au fur et à mesure de la navigation, sans impression de rechargement ; le code JavaScript récupère les données nécessaires sur le serveur grâce à cet objet XMLHttpRequest et construit ensuite dynamiquement l'interface grâce à DOM.

Le fichier XML contenant les informations de configuration

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
  <serveur type="ldap">
    <hote>192.168.0.5</hote>
    <baseDn>o=xulforum,ou=utilisateurs</baseDn>
  </serveur>
  <serveur type="php">
    <hote>localhost</hote>
    <chemin>/~jonathan/xulforum.php</chemin>
  </serveur>
</config>
```

Le fichier XML est très simple. On place un élément racine appelé `config`, puis on définit deux serveurs, avec chacun leurs propres caractéristiques. Nous pourrions parcourir la liste des éléments serveur, voir pour chacun leur type et, en fonction du type de serveur décrit dans le fichier de configuration, en extraire les informations qui nous intéressent.

Récupération de ce fichier XML avec JavaScript

```
function chargerParametres() {
    try {
        var x = new XMLHttpRequest;
        x.open("GET",
            "http://localhost/~jonathan/params.xml", false);

        x.send(null);

        if (x.status != 200) {
            throw new Error("HTTP "+x.status);
        }

    } catch (e) {
        ajouterErreur(
            "Impossible d'obtenir le fichier de configuration !");
        ajouterErreur(e);
        dump(e);
    }

    var config = x.responseXML;
    if (config == null) {
        ajouterErreur("Impossible de lire le fichier XML");
        return;
    }

    gConfig = new configServeur;
    gConfig.xml = x;

    if (gConfig.ldap.hote == null)
        document.getElementById("xf-ident-opt-ignorer").
            ➤ setAttribute("checked", true);
    else
        document.getElementById("xf-ident-opt-ignorer").
            ➤ setAttribute("checked", false);

    document.getElementById("xf-ident-opt-dn").
        ➤ setAttribute("value", gConfig.ldap.baseDn);
}
```

- ◀ Il faut d'abord initialiser l'objet XMLHttpRequest.
- ◀ On demande à ouvrir, par la méthode GET (la méthode classique pour accéder à un document via HTTP, par opposition à POST pour envoyer un formulaire par exemple), le fichier de configuration.
- ◀ On envoie la requête.
- ◀ Si le code de statut HTTP envoyé par le serveur est différent de 200 (*HTTP OK*), on arrête. Quelques codes d'erreur sont très connus : 404 « File not found », 500 « Internal Server Error »...
- ◀ On prend en charge les erreurs, avec un bloc catch. On attrape ainsi les erreurs engendrées par XMLHttpRequest (impossible de se connecter au serveur) et les erreurs lancées avec notre throw.
- ◀ On vérifie que le document XML n'est pas vide.
- ◀ On l'assigne à la variable gConfig que nous allons traiter plus tard.
- ◀ On pré-remplit quelques valeurs, comme la case servant à ignorer l'identification LDAP.
- ◀ On pré-remplit aussi le champ DN de base.

Quelques petits détails sont à remarquer dans ce code.

- L'URL vers le fichier de configuration pourra être remplacée plus tard par `document.getElementById("xf-ident-opt-config").value`.

- Le troisième argument de la méthode `open` indique que la requête ne doit pas être mise en arrière-plan (le paramètre `async` vaut *false*).
- On n'envoie pas de données à la ligne suivante, via `send`. Si la méthode avait été `POST`, on aurait eu une valeur non nulle.

ALTERNATIVE Requête asynchrone

Il est possible de lancer la requête en arrière-plan. Ainsi, le traitement des informations n'est pas bloqué, l'affichage ne donne pas l'impression de se « geler » si le fichier met du temps à arriver (du fait de la taille importante du fichier ou d'une congestion réseau). On pourra ensuite réagir à l'arrivée du fichier XML.

Cependant, cette méthode n'est pas détaillée ici. Elle impose l'introduction des notions d'interfaces implémentées en JavaScript, ce qui rajouterait une complexité à un chapitre déjà dense. De plus, les fichiers sont généralement courts, ce qui limite les risques de gel de l'interface. Enfin, nous verrons les requêtes asynchrones avec LDAP au chapitre 10.

Il y a quelques détails sur le mode *async* sur le wiki du site xulfr, ainsi qu'un très intéressant exemple pour envoyer un fichier par la méthode `POST` (à lire après avoir bien intégré les composants XPCOM en JavaScript).

► <http://xulfr.org/wiki/ApplisWeb/Request>

On mentionne aussi une variable globale `gConfig`. C'est en fait un objet, déclaré de la manière suivante :

La variable `gConfig` qui stockera toutes nos informations de configuration

```
function configServeur() {
    this.php = { hote: null, chemin: null};
    this.ldap = { hote: null, baseDn: null};
    this.xml = null;
    this.toString = function () {
        return "Configuration globale :\n"+
            "\tPHP :\n"+
            "\t\tHote "+this.php.hote+" :: Chemin "+
            this.php.chemin+"\n"+"LDAP :\n"+
            "\t\tHote "+this.ldap.hote+" :: BaseDN "+
            this.ldap.baseDn+"\n";
    }
}
var gConfig;
/* déclarée ici pour pouvoir être globale = accessible depuis
une fonction. La ligne gConfig = new configServeur est ainsi
placée dans la fonction chargerParametres */
```

La variable `gConfig` est un objet qui est créé grâce à sa fonction constructeur `configServeur`. Ses variables membres `php` et `ldap` sont des objets, avec elles-mêmes des variables membres servant à stocker les informations sur les serveurs PHP et LDAP. La variable `xml` contient l'objet

Document créé lors de l'analyse du fichier XML par `XMLHttpRequest`. Elle pourra servir pour des traitements ultérieurs.

La fonction `toString` est quant à elle une fonction un peu particulière. Normalement, chaque instance de la classe `Object` possède une fonction `toString` par défaut, qui renvoie une valeur du type `[object Object]`. C'est cette fonction qui est appelée quand on a besoin de représenter un objet sous sa forme de chaîne. Les objets `Number` ont leur propre fonction `toString` pour convertir un nombre en chaîne ; de même, les tableaux, instances de l'objet `Array`, sont automatiquement convertis en leur suite de valeurs séparées par des virgules lorsqu'on a besoin de les représenter sous forme de chaîne ; enfin, les erreurs renvoient `Error` : suivi du message d'erreur.

Comme notre objet n'appartient à aucune classe particulière, il possède une fonction par défaut qui ne sait bien sûr pas le représenter littéralement. C'est ici que nous intervenons : nous fournissons notre propre fonction `toString` pour offrir une représentation littérale convenable de notre objet `gConfig`. Bien sûr, cela ne sert qu'à des fins de débogage, mais ceci peut être particulièrement utile pour identifier la cause d'un problème en cas d'erreur persistante.

ALTERNATIVES Utilisation de la propriété prototype

La méthode présentée ci-contre pour instancier un objet peut présenter des inconvénients. Si par exemple on avait à instancier plusieurs objets `configServeur`, il faudrait, pour chaque objet, allouer de la mémoire correspondant aux fonctions. Il y aurait ainsi plusieurs emplacements mémoire contenant chacun le code de la fonction `toString`. Cet inconvénient peut être évité en utilisant la syntaxe suivante :

```
function configServeur() {
    this.php = { hote: null, chemin: null};
    this.ldap = { hote: null, baseDn: null};
    this.xml = null;
}

configServeur.prototype.toString =
function () {
    return "Configuration globale :\n"+
        "\tPHP :\n"+
        "\t\tHote "+this.php.hote+
        " :: Chemin "+this.php.chemin+"\n"+
        "\tLDAP :\n"+
```

```
        "\t\tHote "+this.ldap.hote+
        " :: BaseDN "+this.ldap.baseDn+"\n";
};
```

Ainsi, pour tous les nouveaux objets `configServeur`, la variable membre sera juste une référence vers une fonction `toString` définie ailleurs dans le code. Ceci évite de définir deux fois une fonction, en particulier pour du code plus lourd faisant appel à des composants XPCOM.

Déclarer directement l'objet

Dans l'esprit inverse de ce qui précède, on aurait pu dire que, comme on n'utilise qu'une fois l'objet `gConfig`, il n'est pas nécessaire de créer un constructeur : il vaut mieux instancier la variable directement. On aurait ainsi pu écrire :

```
var gConfig = { php : ..., ldap : ...,
    toString : function () { ... } };
```

À vous de retenir la syntaxe qui vous plaît le plus !

Les bases sont posées et nous allons enfin pouvoir agir concrètement : remplir les variables `gConfig.php.hote`, `gConfig.ldap.baseDn`, etc., avec les valeurs qui leur correspondent. DOM va donc de nouveau être mis à contribution, pour parcourir le fichier XML et en extraire les informations qui nous intéressent ; nous remplirons ainsi notre objet de configuration.

L'analyse avec DOM

De même que, lorsque nous montrons les options avancées, nous manipulons une instance globale de l'objet `Document` (appelée `window.document` ou en raccourci `document`), nous allons manipuler une autre instance de cet objet `Document`, qui est `var config=x.responseXML`.

Extraction des informations du fichier XML en JS

```
var serveurs = config.getElementsByTagName("serveur");
for (var i = 0; i < serveurs.length; i++) {
    var s = serveurs[i];
    switch (s.getAttribute("type")) {
        case "php" :
            gConfig.php.hote =
                s.getElementsByTagName("hote")[0].firstChild.data;
            gConfig.php.chemin =
                s.getElementsByTagName("chemin")[0].firstChild.data;
            break;
        case "ldap" :
            gConfig.ldap.hote =
                s.getElementsByTagName("hote")[0].firstChild.data;
            gConfig.ldap.baseDn =
                s.getElementsByTagName("baseDn")[0].firstChild.data;
            break;
    }
}
dump(gConfig);
```

ALTERNATIVE Autre syntaxe

```
gConfig.ldap.baseDn
gConfig["ldap"]["baseDn"]
```

Les syntaxes ci-dessus sont toutes deux équivalentes. On peut préférer la seconde pour remplir dynamiquement un objet (par exemple :

```
gConfig[s.getAttribute("type")]
```

Il faut d'abord récupérer un objet contenant les différents éléments serveur, en l'occurrence un objet `HTMLCollection` (un dump sur cet objet renvoie *Object HTMLCollection*). Comme il ne possède pas que les propriétés 0, 1... pour les différents éléments serveur, mais d'autres propriétés comme `length` (le nombre d'éléments), nous utilisons une boucle `for`. Nous initialisons une variable `i` (traditionnellement un compteur) et tant qu'elle est inférieure au nombre d'éléments serveur (qui vaut 2), nous l'augmentons de un.

À chaque élément serveur, représenté par une variable secondaire `s` (pour éviter les longues récritures), on choisit les actions à effectuer en fonction de l'attribut `type` du tag serveur. Si c'est un serveur PHP, on récupère tous les éléments `hote` du serveur, on prend celui d'indice 0, puis son premier élément (qui est un objet `Text`) et enfin sa valeur avec `data`.

ALTERNATIVE EcmaScript For XML : e4X

► https://bugzilla.mozilla.org/show_bug.cgi?id=293392

À l'heure de l'écriture de ce chapitre, la fonctionnalité n'était pas disponible. Mais pour des versions de Mozilla datées du 5 juillet 2005 (et postérieures), il est possible d'utiliser une nouveauté du langage JavaScript : ECMAScript for XML ou e4X en abrégé.

e4X est une extension à JavaScript (standardisée elle aussi par l'ECMA), qui permet de définir un nouveau type natif : XML. On pourra ainsi écrire :

```
var monxml = <montag>type natif !</montag>;
```

Plus d'informations sur e4x sont disponibles sur le lien ci-dessous. Nous n'utiliserons pas cette technique dans XUL Forum pour de simples raisons de compatibilité descendante !

► <http://weblog.infoworld.com/udell/2004/09/29.html#a1085>

On pourrait détailler tout ceci de la manière suivante :

```
var hotes = s.getElementsByTagName("hote");
// il peut y en avoir plusieurs et c'est la seule méthode
// pour récupérer des éléments selon leur tag
var premierHote = hotes[0];
var noeudTexte = premierHote.firstChild;
var valeurHote = noeudTexte.data;
```

Cependant, tout cela peut être condensé en une ligne, pour gagner en efficacité au niveau du DOM. Nous procédons de même pour les autres attributs des serveurs PHP et LDAP.

Pour conserver l'URL du fichier de configuration entre les pages `xulforum.xul` et `index.xul`, nous utilisons une astuce très simple. Lorsqu'on clique sur le bouton *OK*, l'utilisateur est envoyé non pas vers la page `index.xul`, mais vers la page `index.xul?config=http://www.example.com/fichierdeconf.xml`.

```
//dans xulforum.xul
var url = "chrome://xulforum/content/index.xul?config="+gConfig.url;
document.location.href = url;

//puis dans index.xul
var l = document.location.href;
var c = l.substr(l.indexOf("?config=")+8);
//c contient maintenant l'url du fichier de configuration
```

Tout ceci nous donne de bonnes bases en JavaScript. Dans les chapitres suivants, nous aurons tous les éléments prêts pour passer à quelque chose de plus concret : identification directe au serveur LDAP ou utilisation des services web.

Une seule ombre persiste cependant au tableau : toutes les fonctions, tous les objets sont concentrés dans un seul et unique fichier ! Nous

ATTENTION firstChild trompeur

On pourrait être tenté d'utiliser la syntaxe `s.firstChild.firstChild.data`.

Mais ce serait une erreur car `s.firstChild` est en fait un nœud `Text` contenant le saut de ligne et les tabulations comprises entre la fermeture du premier tag serveur et l'ouverture du tag `host`.

De toute façon, la méthode que nous utilisons permet de se fonder uniquement sur les noms des tags et pas sur leur ordre dans le fichier XML. Si l'on avait utilisé `s.firstChild`, au moindre changement, par exemple, l'inversion des tags `baseDn` et `host`, il aurait fallu mettre à jour le code, ce qui est bien sûr une faute de goût !

ATTENTION**Fonctions à lancer au chargement**

Si vous placez `chargerParametres()` au beau milieu de votre JavaScript, vous obtiendrez des erreurs. L'objet

```
document.getElementById(
    "un-objet-qui-existe")
```

n'aura pas de propriétés. En effet, comme la balise `script` est placée au tout début du fichier XUL, les instructions placées directement dans le JavaScript (et non pas dans une fonction séparée) sont exécutées avant que tous les éléments ne soient chargés. Le premier appel à `chargerParametres()` doit se trouver dans la fonction `initialisation()` vue page 113. Ainsi, elle sera appelée après le chargement complet du fichier XUL.

ASTUCE Rechargement d'un fichier de configuration

Maintenant que notre fonction `chargerParametres` est prête, nous pouvons proposer le chargement d'un fichier de configuration alternatif !

Il suffit de placer le code suivant dans le fichier XUL :

```
<row>
  <hbox>
    <description value="&ident.ident_opt_config;" />
    <button label="Recharger" oncommand="chargerParametres()" />
  </hbox>
  <textbox id="xf-ident-opt-config"
    value="http://localhost/~jonathan/params.xml" />
</row>
```

Ainsi, comme la fonction `chargerParametres` récupère la valeur du champ contenant l'URL du fichier de configuration, après avoir mis à jour ce champ, l'utilisateur n'aura plus qu'à cliquer sur *Recharger* pour prendre en compte les nouveaux paramètres (comme le DN de base).

pourrions séparer le tout en plusieurs fichiers et les inclure avec chacun une balise `script`, mais nous allons faire mieux que cela.

Approche des composants XPCOM : fonction `include()`

Si vous avez l'expérience d'autres langages, vous avez certainement déjà utilisé du code comme `#include <headers_dans_autre_fichier.h>` ou `include ("../include/routines.inc.php")`.

JavaScript ne propose pas ce mécanisme par défaut, mais il existe un composant XPCOM qui propose une fonction équivalente. Les composants XPCOM sont écrits en C++ et, grâce à des fichiers IDL (Interface Definition Language), ils proposent, entre autres langages, un accès à JavaScript. Nous avons déjà abordé la place d'XPCOM au sein du XPFE dans le chapitre d'introduction.

Les composants XPCOM

Concrètement, comme ce mécanisme fonctionne-t-il en JavaScript ? Il faut d'abord obtenir une instance d'un composant. C'est-à-dire que, parmi tous les composants disponibles, on en choisit un en particulier et on l'initialise : une variable est créée, qui contient une instance de ce composant. Mais cela ne suffit pas pour utiliser le composant directement. Un composant implémente plusieurs interfaces, décrites dans des

fichiers IDL. Nous devons demander à l'instance du composant de nous fournir une interface, afin que nous puissions communiquer avec lui.

En résumé, il faut :

- d'abord créer une instance du composant ;
- ensuite, lui demander une interface via laquelle communiquer avec lui ;
- on peut alors utiliser les méthodes et les propriétés fournies par l'interface sur ce composant.

Il existe ainsi un composant permettant de gérer un téléchargement. Il implémente trois interfaces. La première est standard et est implémentée par tout composant XPCOM. La seconde concerne le téléchargement en lui-même et donne accès à l'heure de début, à la source, au fichier de destination, etc. La troisième est plus spécifique à la progression : elle permet de lier le téléchargement à d'autres fonctions chargées de mettre à jour l'interface. Cette troisième interface est elle aussi implémentée par d'autres composants : envoi d'un e-mail, impression...

Notre inclusion

Il suffit maintenant de placer toutes les routines générales, comme la gestion des erreurs, dans le fichier `content/javascript/global.js`. Pour l'inclure, on écrira dans le fichier `identification.js` :

Inclusion dans `identification.js`

```
var jComposant = Components.classes[
    "@mozilla.org/moz/jssubscript-loader;1"].createInstance();
var jInterface = jComposant.QueryInterface(
    Components.interfaces.mozIJSSubScriptLoader);
jInterface.loadSubScript(
    "chrome://xulforum/content/javascript/global.js");
```

Nous pourrions recopier ces trois lignes dans le futur `forum.js`, qui contiendra le JavaScript correspondant au fichier `index.xul`. Le mécanisme d'inclusion pourra être repris pour différentes pages, chacune nécessitant des inclusions différentes (la page `index.xul` aura besoin des routines concernant les services web et la page `xulforum.xul` des routines LDAP), et aussi pour inclure différents fichiers. Ainsi on ajoutera plus tard :

```
jInterface.loadSubScript(
    "chrome://xulforum/content/javascript/ldap.js");
```


B.A.-BA Contract ID

Les composants XPCOM sont identifiés par une chaîne de caractères commençant par @. C'est l'identifiant unique qui caractérise un composant XPCOM. Il est appelé « Contract ID » et doit être lisible par un humain. Il se compose des éléments suivants :

- le domaine du composant : mozilla.org pour les composants livrés d'office, xulforum.org si nous enregistrons nos propres composants, etc. ;
- le module : il peut s'agir de network, ldap, gfx, core, etc. ;
- le composant : nous en verrons de nombreux dans la suite de ce livre ; dans ce cas précis, le composant est jssubscript-loader ;
- la version du composant.

Les trois premiers éléments sont séparés par des barres obliques (slash) ; le dernier est séparé du reste par un point-virgule.

Les fonctions se rapportant à la configuration de XUL Forum seront placées dans un fichier séparé `config.js`, tandis que les routines de gestion d'erreurs le seront dans `global.js`. Le déplacement des fonctions est transparent avec cette méthode d'inclusion.

Comme expliqué précédemment, la première ligne crée une instance du composant : on accède à l'objet global `Components`, à son membre `classes`, puis au membre `@mozilla.org/moz/jssubscript-loader;1` (remarquez l'utilisation de la syntaxe avec guillemets, obligatoire pour ne pas causer d'erreurs avec les caractères spéciaux), puis à sa méthode `createInstance()`.

Ensuite, on demande à accéder à l'interface `mozIJSSubscriptLoader`. Une fois que l'on a récupéré cette interface, on s'en sert pour utiliser les méthodes du composant, en l'occurrence la méthode `loadSubScript`, avec pour chemin une URL `chrome://`.

Le composant

- ▶ http://www.xulplanet.com/references/xpcomref/comps/c_mozjssubscriptloader1.html

L'interface

- ▶ <http://www.xulplanet.com/references/xpcomref/ifaces/mozIJSSubScriptLoader.html>

Nous aurons l'occasion de revenir largement sur les composants XPCOM dans les chapitres suivants sur JavaScript, ceci n'est qu'un bref aperçu de la logique de composants.

CULTURE NSPR et JavaScript

Si vous étudiez de près l'interface, vous remarquerez que les entiers ne sont pas de type `int` mais `PRUint32`. PR signifie ici « Portable Runtime », un signe de la couche d'abstraction NSPR (Netscape Portable Runtime) qui sous-tend tout l'architecture XPCOM et qui garantit que le type de données `PRUint32` aura la même taille sur toutes les plateformes.

En résumé...

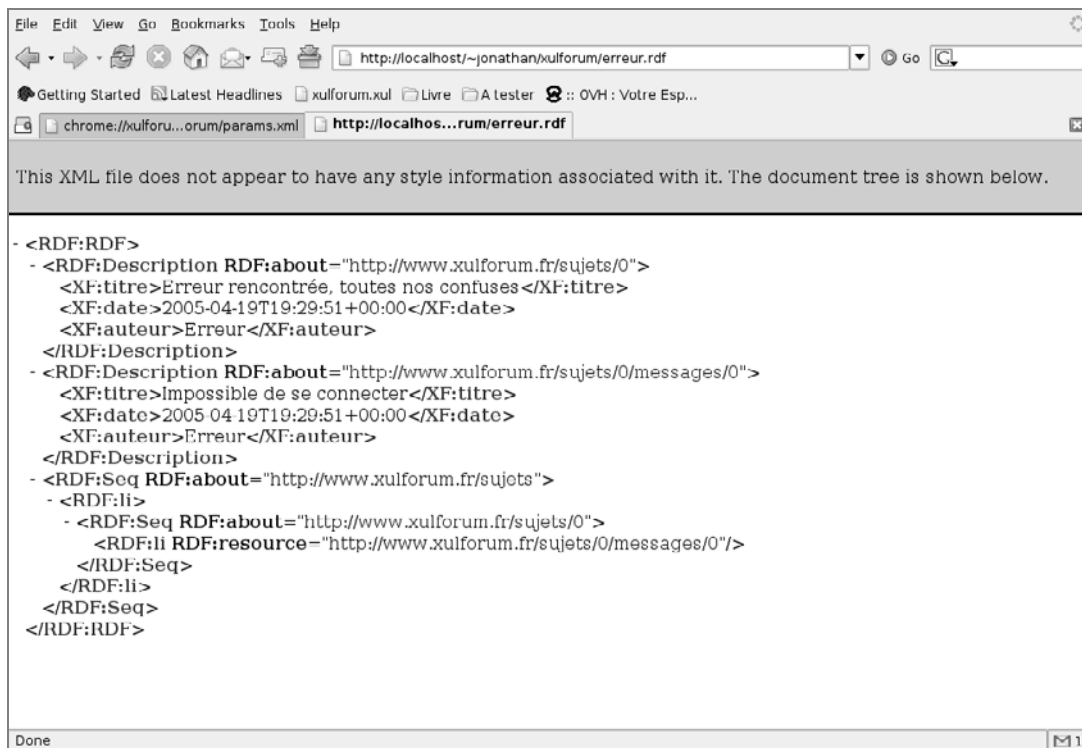
Beaucoup d'avancées concrètes ont été effectuées avec ce chapitre :

- Tout d'abord, l'interface n'est plus statique : du JavaScript est désormais lié à notre fichier `xulforum.xul` et permet d'effectuer de manière dynamique des modifications dans l'interface (par exemple, montrer les options avancées), grâce au DOM.
- Les erreurs ont leur propre mécanisme d'affichage, dans la barre de statut, comme toute application qui se respecte.
- L'internationalisation n'a pas été oubliée : le JavaScript est lui aussi multi-langue !
- La communication avec l'extérieur a été vue dans un sens : du serveur vers le client, pour rapatrier un fichier de configuration XML, manipulé grâce au DOM, rapatrié grâce au fameux `XMLHttpRequest`.
- Un mécanisme d'inclusion de fichiers JavaScript nous a permis d'effectuer une première approche des composants XPCOM de Mozilla.

Tout cela est très dense, c'est pourquoi un chapitre sur RDF remplira le rôle d'intermède : en créant des fichiers XML avec PHP, nous allons exposer le contenu de la base de données contenant les messages et, en utilisant un mécanisme de modèles (*templates* en anglais) au sein de XUL, nous allons créer automatiquement du contenu.

8

chapitre



Automatisation avec RDF

Pour remplir notre interface, nous pourrions utiliser JavaScript, nous connecter au serveur distant, récupérer des données, les analyser, utiliser DOM, remplir l'interface, etc. Mais quelle complexité ! Alors qu'avec RDF, tout est automatique...

SOMMAIRE

- ▶ Préliminaires : format RDF, DOM côté PHP
- ▶ Une approche linéaire, qui devient récursive
- ▶ Plus de contrôle sur RDF avec XPCOM

MOTS-CLÉS

- ▶ Templates
- ▶ Service RDF XPCOM
- ▶ RDF/XML

Nous allons maintenant introduire un nouveau standard, au cœur de la plate-forme Mozilla : RDF. Les concepts fondamentaux vus lors du chapitre de présentation du XPFE vont être approfondis. Nous utiliserons dans ce chapitre deux méthodes : l'une avec JavaScript et l'autre sans.

Le format RDF : explications détaillées

Les nœuds et les arcs

ALTERNATIVE Utilisation d'autres méthodes de sérialisation

Il existe aussi d'autres langages comme N-Triples permettant de modéliser un schéma RDF. Cependant, seul RDF/XML est implémenté au sein de Mozilla, c'est donc cette solution que nous retiendrons. D'autant plus que XML devient maintenant familier. Un dérivé supplémentaire ne devrait pas être trop difficile à intégrer !

RDF signifie *Resource Description Framework*. C'est au départ un format graphique : il représente les données sous forme de schéma structural. Mais on ne peut pas inclure des schémas dans un langage de programmation ! Il existe donc des formats pour sérialiser (représenter sous forme de texte) les schémas RDF : c'est le rôle du format RDF/XML.

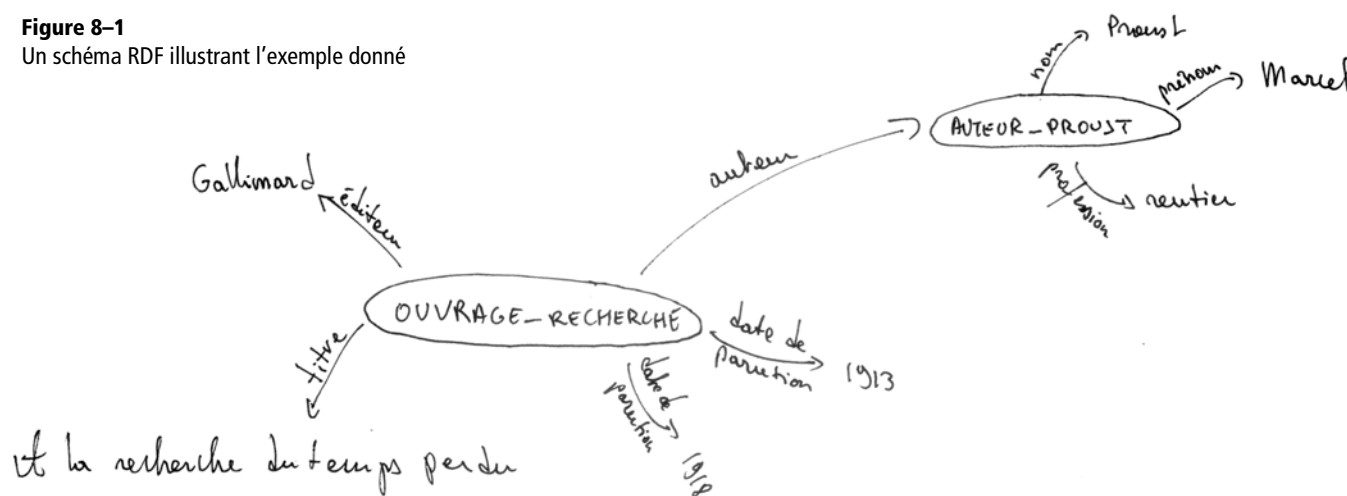
Le plus souvent, lorsqu'on parle d'un fichier RDF, par abus de langage on désigne un fichier RDF/XML.

Dans un graphe RDF, il y a des nœuds et des arcs. Les nœuds vont représenter des entités, comme le livre *À la recherche du temps perdu*, comme un lecteur de ce livre, ou encore comme son auteur...

Les arcs vont relier les nœuds entre eux. Par exemple l'arc « auteur » relie le nœud « À la recherche du temps perdu » à un autre nœud « Marcel Proust ». Un autre arc « date de parution » peut pointer vers 1913. On peut aussi faire pointer un second arc « date de parution » vers 1918 (la parution de l'ouvrage a été échelonnée).

Figure 8-1

Un schéma RDF illustrant l'exemple donné



RDF permet donc de représenter des relations logiques entre différents éléments, pour permettre à un ordinateur de se construire un arbre, où les différents nœuds sont reliés entre eux et où les relations sont logiques : relations de livre à auteur pour un ouvrage, de parents à enfants dans une famille, etc.

RDF est l'un des standards s'inscrivant dans l'initiative globale du Web sémantique du W3C.

En effet, si, au départ, les informations sont le plus souvent structurées (dans une base de données par exemple), les outils web actuels ne retiennent aucun élément du modèle de données, pour se concentrer exclusivement sur la forme. Il devient alors impossible de mettre en œuvre un traitement intelligent de l'information (par un moteur de recherche par exemple). Seul l'être humain est en mesure de retrouver les relations.

RDF est l'un des moyens proposés par le W3C pour modifier cet état de fait en permettant de décrire les relations logiques et structurées entre différents éléments.

► <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>

CULTURE **Formats RSS et RDF**

Depuis quelques années, notamment grâce au développement des blogs, de nombreux sites proposent au lecteur de suivre directement leurs dernières nouveautés grâce à un fichier RSS. Il existe différentes versions entièrement distinctes de cette norme : RSS 0.91, RSS 2.0, RSS 1.0. Les versions 0.91 et 2.0 sont des fichiers XML traditionnels et la version 1.0 est fondée sur RDF.

En couplant un format standard à l'API RDF de Mozilla, on obtient facilement une intégration réussie : c'est ainsi que vous pouvez ajouter dans vos favoris un fil RSS, qui sera par exemple intégré dans Firefox, sous forme de liste contenant les différentes nouvelles.

Nœuds et URI

Vous pouvez utiliser trois types différents de nœuds :

- Nœuds littéraux : c'est ce que nous avons utilisé plus haut, ils contiennent l'équivalent de chaînes en programmation. Exemple : « Marcel Proust »
- Nœuds URI : lorsque vous faites référence à un nœud qui a été défini ailleurs, ou à un nœud qui a lui aussi ses propres propriétés, c'est en quelque sorte un objet. Ainsi, dans l'exemple simplifié précédent, « À la recherche du temps perdu » serait le nœud <http://www.xulforum.org/livres/recherche>, auquel se lieront des arcs pour définir différentes relations logiques du livre : vers son auteur, ses 7 tomes, etc.
- Nœuds vides : lorsque vous faites référence à un nœud URI qui n'a pas encore été défini.

Pour éviter de représenter sous forme de texte les relations décrites précédemment (même si cela est possible avec N-Triples par exemple), un schéma sera largement plus clair !

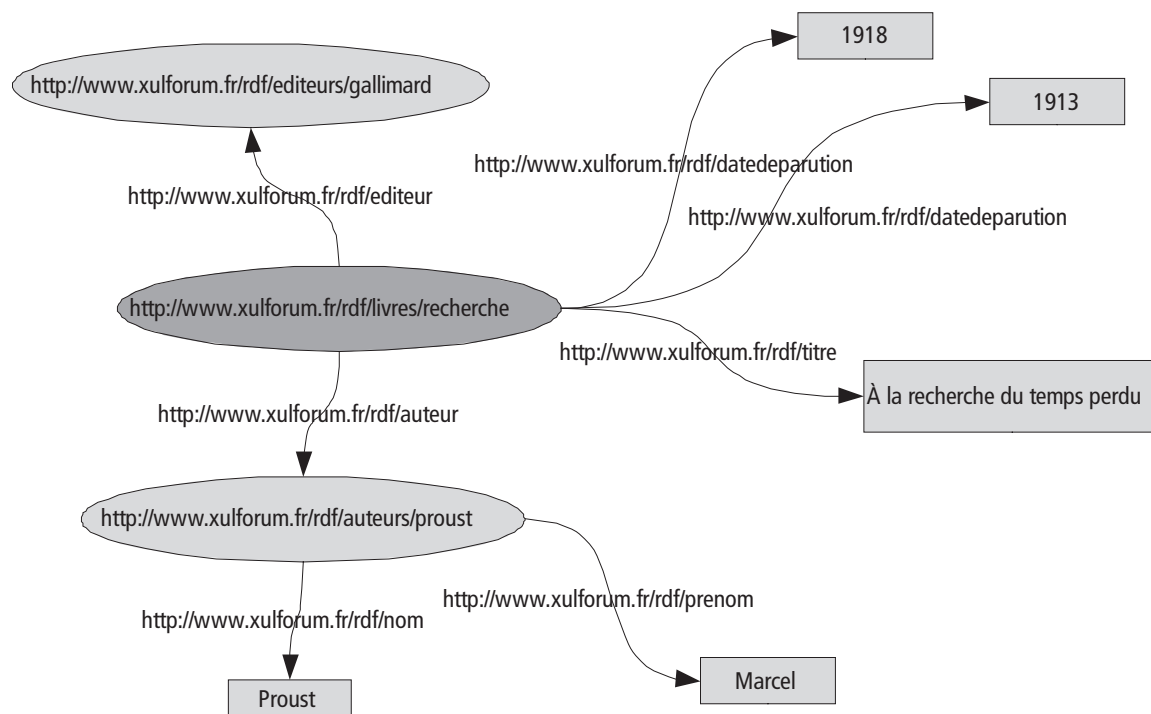


Figure 8-2 Le schéma RDF incluant les relations telles que nous les avons décrites

Sérialisation de RDF avec RDF/XML

La sérialisation est très simple : nous placerons à l'élément racine deux espaces de nommage : l'un pour RDF/XML et l'autre pour nos propres éléments (ce sera `http://www.xulforum.org/rdf#`). En fait, l'espace de nommage choisi n'a aucune relation avec un site web. Simplement, pour éviter les confusions avec d'autres sites, nous utilisons un espace de nommage fondé sur l'adresse du site. Le dièse final est une convention qui permet d'écrire les champs sous la forme `http://www.xulforum.org/rdf#nomduchamp` et les nœuds sous la forme `http://www.xulforum.org/rdf/noeud1/noeud2`. Dans les fichiers `contents.rdf`, nous n'utilisons pas d'URI fondées sur un site web, mais des URI commençant par `urn:mozilla`. Le résultat est bien sûr le même.

En RDF/XML, on utilisera un élément `Description` pour décrire un nœud.

```
<?xml version="1.0" ?> ❶
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:XF="http://www.xulforum.org/rdf#"> ❷
  <RDF:Description
    RDF:About="http://www.xulforum.org/rdf/livres/recherche"
    XF:titre="A la recherche du temps perdu" /> ❸
  </RDF:RDF>
```

D'abord, on ouvre le fichier avec le désormais traditionnel prologue XML ❶. Ensuite, on spécifie l'élément racine `RDF:RDF`, avec comme espace de nommage celui correspondant au standard RDF ❷.

L'élément intéressant est l'élément `RDF:Description` ❸. Il représente un nœud, le nœud `http://www.xulforum.org/rdf/livres/recherche`, spécifié grâce à l'attribut `RDF:About`. Ses attributs ne dépendent pas du format RDF/XML mais de notre contenu, ils utilisent donc l'espace de nommage `XF`, comme XUL Forum !

Via l'attribut `XF:titre`, remplacé grâce à l'espace de nommage par l'arc `www.xulforum.org/rdf#titre`, on le relie à un nœud texte représentant son titre. Cependant, deux problèmes se posent.

- Si l'on veut spécifier un autre nœud URI pointant vers l'auteur par exemple, l'utilisation d'un attribut n'est pas possible. En effet, comment faire rentrer un nouvel élément `Description` dans un attribut ?
- Si l'on veut spécifier deux valeurs pour les dates de parutions, ceci ne sera pas possible, car on ne peut spécifier deux fois un attribut en XML.

Nous allons donc utiliser une syntaxe alternative, équivalente à celle vue plus haut, qui nous permettra de spécifier deux valeurs pour la date de parution et un nœud pour l'auteur.

```
<?xml version="1.0" ?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:XF="http://www.xulforum.org/rdf#">
  <RDF:Description
    RDF:About="http://www.xulforum.org/rdf/auteurs/proust"
    XF:prenom="Marcel" XF:nom="Proust" />
  <RDF:Description
    RDF:About="http://www.xulforum.org/rdf/livres/recherche"
    XF:titre="A la recherche du temps perdu">
    <XF:datedeparution>1918</XF:datedeparution>
    <XF:datedeparution>1913</XF:datedeparution>
    <XF:auteur
      RDF:Resource="http://www.xulforum.org/rdf/auteurs/proust" />
    </RDF:Description>
  </RDF:RDF>
```


On crée un premier nœud appelé auteurs/proust, avec nom et prenom. Ensuite, dans le nœud livres/recherche, on ajoute trois éléments fils. Deux dates de parution, c'est-à-dire deux éléments XF:datedeparution fils et un élément XF:auteur, pointant vers un autre nœud grâce à l'utilisation de l'attribut RDF:Resource (c'est ici un lien entre éléments qui est spécifié dans le cadre de RDF/XML).

Les deux syntaxes peuvent être mêlées pour simplifier la lisibilité. La première est recommandée car elle est moins gourmande en balises supplémentaires, mais la seconde est souvent indispensable.

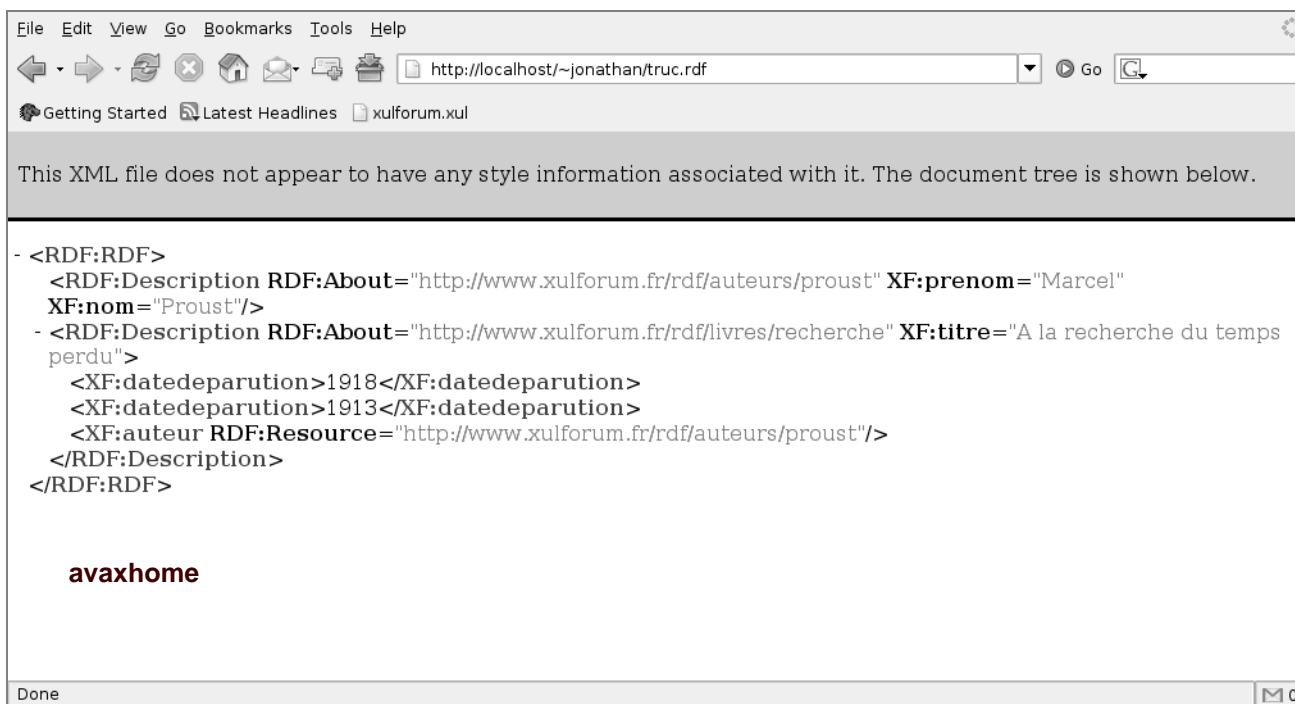


Figure 8-3 Notre document RDF visualisé dans Mozilla

OUTILS Configuration du serveur Apache

Apache ne sait pas comment gérer les fichiers RDF par défaut et les envoie au navigateur du visiteur comme de simples fichiers texte. Ceci empêche Mozilla d'utiliser son parseur XML pour vous permettre de détecter d'éventuelles erreurs de syntaxe. Il faut éditer sur les serveurs web un fichier appelé `.htaccess`, qui sera placé dans le dossier contenant les fichiers RDF. Ajoutez-lui la ligne suivante :

```
AddType text/xml .rdf
```

Ainsi, vous pourrez placer votre fichier RDF sur un serveur web pour le visualiser directement dans Mozilla et tirer parti de son analyse XML. L'utilisation d'une « URL file » ne marche pas, Mozilla tentant d'interpréter le fichier RDF comme une page web classique.

Listes

Pour utiliser une liste en RDF, on utilisera des arcs spéciaux définis par RDF : `_1` pour l'arc menant vers le premier élément, `_2` et ainsi de suite. L'élément `RDF:Description` sera remplacé par `RDF:Seq`, pour annoncer une séquence ordonnée d'éléments, car il faut prendre en compte l'ordre de parution des différents tomes.

Ainsi, on pourra écrire, pour lister les différents tomes de la recherche :

```
<RDF:Seq
  RDF:About="http://www.xulforum.org/rdf/livres/recherche"
  <RDF:_1>
    <RDF:Description
      RDF:About="http://www.xulforum.org/rdf/livres/recherche1"
      XF:titre="Du côté de chez Swann" />
    </RDF:_1>
  <RDF:_2>
    <RDF:Description
      RDF:About="http://www.xulforum.org/rdf/livres/recherche2"
      XF:titre="A l'ombre des jeunes filles en fleurs" />
    </RDF:_2>
  ...
</RDF:Seq>
```

Cependant, ceci n'est pas aisé à gérer. RDF/XML nous propose donc l'utilisation d'un élément spécial `li` qui numérote automatiquement. Le code suivant est donc équivalent :

```
<RDF:Seq
  RDF:About="http://www.xulforum.org/rdf/livres/recherche"
  <RDF:li>
    <RDF:Description
      RDF:About="http://www.xulforum.org/rdf/livres/recherche1"
      XF:titre="Du côté de chez Swann" />
    </RDF:li>
  <RDF:li>
    <RDF:Description
      RDF:About="http://www.xulforum.org/rdf/livres/recherche2"
      XF:titre="A l'ombre des jeunes filles en fleurs" />
    </RDF:li>
  ...
</RDF:Seq>
```

C'est le principe utilisé dans les fichiers `contents.rdf` de Mozilla : le nœud `urn:mozilla:package:root` est une liste contenant les différents paquets décrits, le principal étant bien sûr `urn:mozilla:package:xulforum`. Mais on aurait pu ajouter `urn:mozilla:package:xfadmin` pour une interface d'administration au forum ; notre extension aurait alors contenu plusieurs paquets.

CULTURE Les différents conteneurs RDF

RDF : Seq fait partie des conteneurs RDF ; ils sont au nombre de trois, et tous se remplissent le plus souvent avec des éléments RDF : `li`. On retrouvera ainsi RDF : Bag pour contenir des éléments non ordonnés, et RDF : Alt, où le premier RDF : `li` représente une valeur par défaut et les suivants, des alternatives.

► <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/#section-Syntax-list-elements>

ASTUCE Vérifiez vos fichiers RDF !

Vous pouvez d'abord les ouvrir via Mozilla, pour vérifier que le document n'est pas mal formé et ensuite les soumettre au validateur du w3c.

► <http://www.w3.org/RDF/Validator/>

Génération de RDF avec PHP

Objectifs

Nous allons utiliser PHP 5 pour nous connecter à une base de données MySQL contenant les messages du forum. Le script PHP devra se connecter au serveur MySQL, utiliser le DOM pour créer le document RDF et l'envoyer sur le navigateur du client. Ensuite, nous utiliserons XUL pour créer du contenu fondé sur ce fichier RDF.

La structure des tables est la suivante :

```
mysql> show columns from xf_sujets;
Field      Type      Null      Key      Default      Extra
id         int(7)    NULL      PRI      NULL         auto_increment
titre      varchar(255)
texte      text
auteur     varchar(255)
date       datetime          0000-00-00

mysql> show columns from xf_messages;
Field      Type      Null      Key      Default      Extra
id         int(7)    NULL      PRI      NULL         auto_increment
s_id      int(7)    NULL      NULL      0
titre      varchar(255)
texte      text
auteur     varchar(255)
date       datetime          0000-00-00
```

ALTERNATIVE SQLite

Une autre nouveauté très importante de PHP5 est l'inclusion de la base de données SQLite. Elle se passe totalement de serveur, en écrivant directement dans un fichier le contenu des tables. Il n'y a donc qu'une librairie cliente. Il aurait été intéressant d'utiliser cette nouvelle technologie pour XUL Forum, mais là encore, les mêmes limitations que pour MySQLi refont surface.

📖 *Les cahiers du programmeur PHP5*, Eyrolles (chapitre sur SQLite)

Deux tables sont utilisées. L'une pour stocker les sujets, avec cinq champs : un identifiant (un entier à sept chiffres), le titre (255 caractères maximum), le texte, l'auteur (même taille) et la date (un champ sous la forme YYYY-MM-DD HH:MM:SS) du sujet. La seconde stocke les messages : le champ `s_id` correspond à l'identifiant du sujet auquel se rapporte ce message.

L'utilisation de PHP5 nous permettra de tirer parti de son nouveau modèle DOM, plus conforme à l'implémentation standard que celui de son prédécesseur PHP4 et de son modèle objet, qui tous deux permettront d'accentuer le parallèle avec le travail déjà effectué en JavaScript dans les chapitres précédents.

INDISPENSABLE La référence DOM de PHP

Un bon développeur PHP ne connaît pas les fonctions par cœur : il sait utiliser efficacement la documentation. C'est pourquoi, vous aurez souvent une fenêtre ouverte à l'adresse suivante :

► <http://fr.php.net/DOM>

B.A.-BA **PHP**

PHP est un langage de script créé au milieu des années 1990 par Rasmus Lerdorf. Au départ constitué de quelques scripts Perl, PHP a été réécrit en C, puis s’est vu ajouter un nouveau moteur pour sa version 3. Les nombreuses extensions, la facilité de mise en œuvre, la syntaxe familière au développeur l’ont propulsé comme premier langage de développement web avec sa version 4. Encore largement en usage à l’heure actuelle, cette version est appelée, à terme, à disparaître au profit de la nouvelle version, PHP5. Cette version apporte ce que les développeurs attendaient depuis longtemps : un modèle objet enfin robuste et fiable, une interaction accrue grâce à une extension SOAP, un support des exceptions, etc.

Sa syntaxe est extrêmement simple et se rapproche d’un langage comme le C :

- les noms de variables sont précédés du signe dollar \$;
- les méthodes d’un objet sont appelées grâce à l’opération ->, familière à ceux ayant manipulé des pointeurs de structures en C ;
- les instructions se terminent là encore toutes par un point-virgule ;
- les blocs try/catch fonctionnent de la même manière qu’en JavaScript ;
- le code est encadré dans la page HTML entre des balises <?php et ?> ;
- les chaînes se concatènent grâce à l’opérateur . (point).

Le fichier PHP

La syntaxe utilisée est orientée procédural pour MySQL, car l’extension est originaire de la version 4 de PHP, version où le modèle objet n’était pas très prononcé. En revanche, la nouvelle extension DOM de PHP5 est entièrement objet.

La génération d’un fichier RDF en PHP5

```
<?php
header("Content-type: text/xml");

$d = new DOMDocument('1.0', 'utf-8');

$r = $d->createElementNS(
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#", "RDF:RDF");

$r->setAttribute("xmlns:XF", "http://www.xulforum.org/rdf#");

$d->appendChild($r);

try {
```

- ◀ Pré-requis : un en-tête HTTP indiquant que l’on envoie un fichier XML au navigateur.
- ◀ La variable d, comme document, représente le document DOM que nous manipulons.
- ◀ On commence par créer l’élément RDF:RDF racine, via la méthode createElementNS (le premier argument est le nom de l’espace de nommage associé, le second est le nom de l’élément).
- ◀ On choisit un second espace de nommage pour l’élément racine.
- ◀ On ajoute l’élément racine en premier dans le document DOM.
- ◀ Nous entrons dans un bloc try/catch, d’où l’on sortira à la première erreur.

On tente de se connecter au serveur MySQL. Si l'on échoue, on lance une exception. Le @ supprime les erreurs lancées sur la sortie standard qui compromettraient l'analyse du document XML.

De même, on sélectionne la base de données qui contient nos tables. En cas d'erreur, nouvelle exception (remarque : `mysql_select_db` est équivalent).

La requête SQL demande à rapatrier quatre colonnes, dans l'ordre descendant des dates (message le plus récent en premier).

On initialise un tableau qui contient la liste des id des sujets.

On lance une boucle qui, tant qu'il y a des résultats à lire en provenance de la requête MySQL, crée un élément `RDF:Description`. Ensuite, on ajoute un attribut `about`, qui contiendra une valeur similaire à : `http://www.xulforum.org/sujets/[[id du sujet]]` avec un remplissage du tableau `idsSujets` pour ensuite identifier les sujets lorsque nous en ferons la liste plus tard. La boucle remplit aussi divers attributs concernant le sujet (titre, auteur, date) puis l'ajoute à l'élément racine.

L'étape suivante crée l'élément `RDF:Seq` amené à contenir la liste des sujets.

On parcourt le tableau contenant les id de tous les sujets existants. Pour chaque élément, on récupère l'id correspondant, on crée une ressource `http://www.xulforum.org/sujets/[[id du sujet]]` et on l'ajoute à l'élément `RDF:Seq`.

Enfin, on ferme le bloc `try` et on attrape une éventuelle exception. La dernière étape exporte l'arbre `$d` en XML et l'affiche sur la sortie standard, c'est-à-dire la page web.

```
$mysqlConn = @mysql_connect("192.168.0.5", "jonathan",
                             "****");
if (!$mysqlConn)
    throw new Exception("Impossible de se connecter !");

$mysqlRessource = @mysql_query("USE jonathan");
if (!$mysqlRessource)
    throw new Exception("Impossible de sélectionner la base");

$mysqlRessource = @mysql_query(
    "SELECT titre,auteur,date,id FROM xf_sujets
    ORDER BY date DESC");
if (!$mysqlRessource)
    throw new Exception("Impossible de chercher");

$idsSujets=array();

while ($o = mysql_fetch_object($mysqlRessource)) {
    $e = $d->createElement("RDF:Description");
    $e->setAttribute("RDF:about",
        "http://www.xulforum.org/sujets/".$o->id);
    $e->appendChild($d->createElement("XF:titre",
        utf8_encode($o->titre)));
    $e->appendChild($d->createElement("XF:date",
        utf8_encode($o->date)));
    $e->appendChild($d->createElement("XF:auteur",
        utf8_encode($o->auteur)));
    $r->appendChild($e);
    $idsSujets[] = $o->id;
}

$e = $d->createElement("RDF:Seq");
$e->setAttribute("RDF:about",
    "http://www.xulforum.org/sujets");
$r->appendChild($e);

foreach ($idsSujets as $id) {
    $l = $d->createElement("RDF:li");
    $l->setAttribute("RDF:resource",
        "http://www.xulforum.org/sujets/$id");
    $e->appendChild($l);
}

} catch (Exception $e) {
    exit($e->getMessage());
}

echo $d->saveXML();
?>
```

POUR ALLER PLUS LOIN MySQLi

MySQLi, comme « MySQL improved », est la nouvelle extension MySQL de PHP. Elle permet de tirer profit des améliorations présentes dans les versions supérieures à 4.1 de MySQL et introduit un réel modèle objet.

Nous avons volontairement choisi de ne pas utiliser cette extension, parce que la plupart des lecteurs ayant déjà abordé PHP connaissent surtout MySQL et que, MySQLi, encore récente, n'est pas encore entrée dans les mœurs chez les hébergeurs. Si vous voulez utiliser PHP4 (votre hébergeur vous l'impose et n'est pas encore passé à PHP5), vous n'aurez en fait que la partie DOM à « backporter ».

Si vous voulez aller plus loin, la réécriture du code en utilisant l'extension MySQLi et son modèle objet serait un bon exercice !

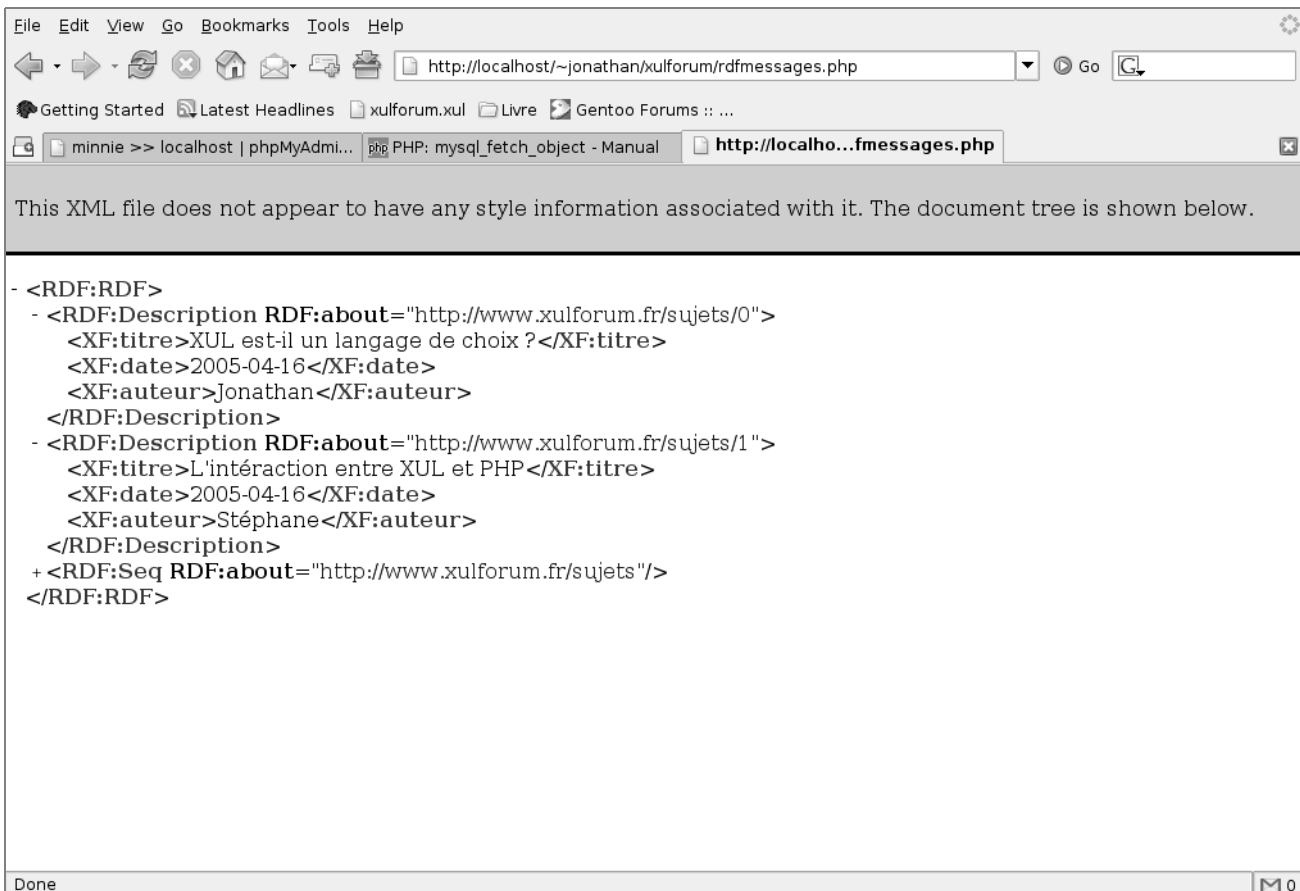


Figure 8-4 Le document RDF créé par le script PHP

ATTENTION Mise à jour des sources RDF

Par défaut, Mozilla met plusieurs minutes avant de remettre à jour une source RDF (placée en cache auparavant). Nous verrons plus loin dans ce chapitre comment contrôler la mise à jour d'une source avec JavaScript ; pour l'instant, la meilleure solution reste de fermer/ouvrir le navigateur.

ATTENTION Liste limitée

Comme nous utilisons PHP, une précaution intéressante serait de limiter, pour cette création en particulier, le nombre de sujets fournis : on pourra ainsi utiliser une URL comme `http://.../rdfmessages.php?nombre=5` pour restreindre du côté de PHP le nombre de sujets à fournir.

Retour à XUL

Maintenant que le fichier RDF contenant les données est créé, XUL va pouvoir en tirer immédiatement profit. Un petit cadre présentant les derniers messages du forum en page d'accueil nous permettra d'informer l'utilisateur et de nous former aux modèles (templates) XUL.

Un premier modèle simple

Le premier modèle XUL

```
<vbox datasources=
  "http://localhost/~jonathan/xulforum/rdfmessages.php"
  ref="http://www.xulforum.org/sujets"> ❶
  <label style="font-weight: bold;"
    value="Les derniers sujets sur Xul Forum !" />
  <template> ❷
    <rule> ❸
      <conditions> ❹
        <content uri="?sujets" /> ❺
        <member container="?sujets" child="?sujet" /> ❻
        <triple subject="?sujet"
          predicate="http://www.xulforum.org/rdf#titre"
          object="?titre" /> ❼
      </conditions>
      <action> ❽
        <label uri="?sujet" value="?titre" /> ❾
      </action>
    </rule>
  </template>
</vbox>
```

La première étape ❶ consiste à ouvrir une boîte. On indique d'abord qu'elle est amenée à être remplie par le contenu fourni par le fichier RDF `rdfmessages.php`, grâce à son attribut `datasources` et on indique aussi que l'élément RDF de base sera `http://www.xulforum.org/sujets`, c'est-à-dire la séquence des sujets. L'élément `template` en ❷ annonce l'ouverture d'une partie décrivant la manière de créer le code. L'élément `rule` ❸ est un conteneur pour les différentes règles de génération. Il contient en premier les conditions ❹, qui se structurent de la manière suivante :

- l'élément `content` définit une variable `?sujets` ❺ qui représente la séquence d'éléments `http://www.xulforum.org/sujets`, sur laquelle nous nous sommes fondés pour notre création de contenu ;
- on parcourt la séquence `?sujets` ❻ et chaque élément de cette séquence se voit associer un nom de variable : `?sujet` (sans s !) ;
- pour chaque `?sujet` ❼, on associe à sa propriété `http://www.xulforum.org/rdf#titre` la variable `?titre`.

Et une fois que les conditions sont définies, il est temps de passer à l'action ❸ ! On crée un élément `label` montrant le titre du sujet ❹. Le résultat est le suivant :

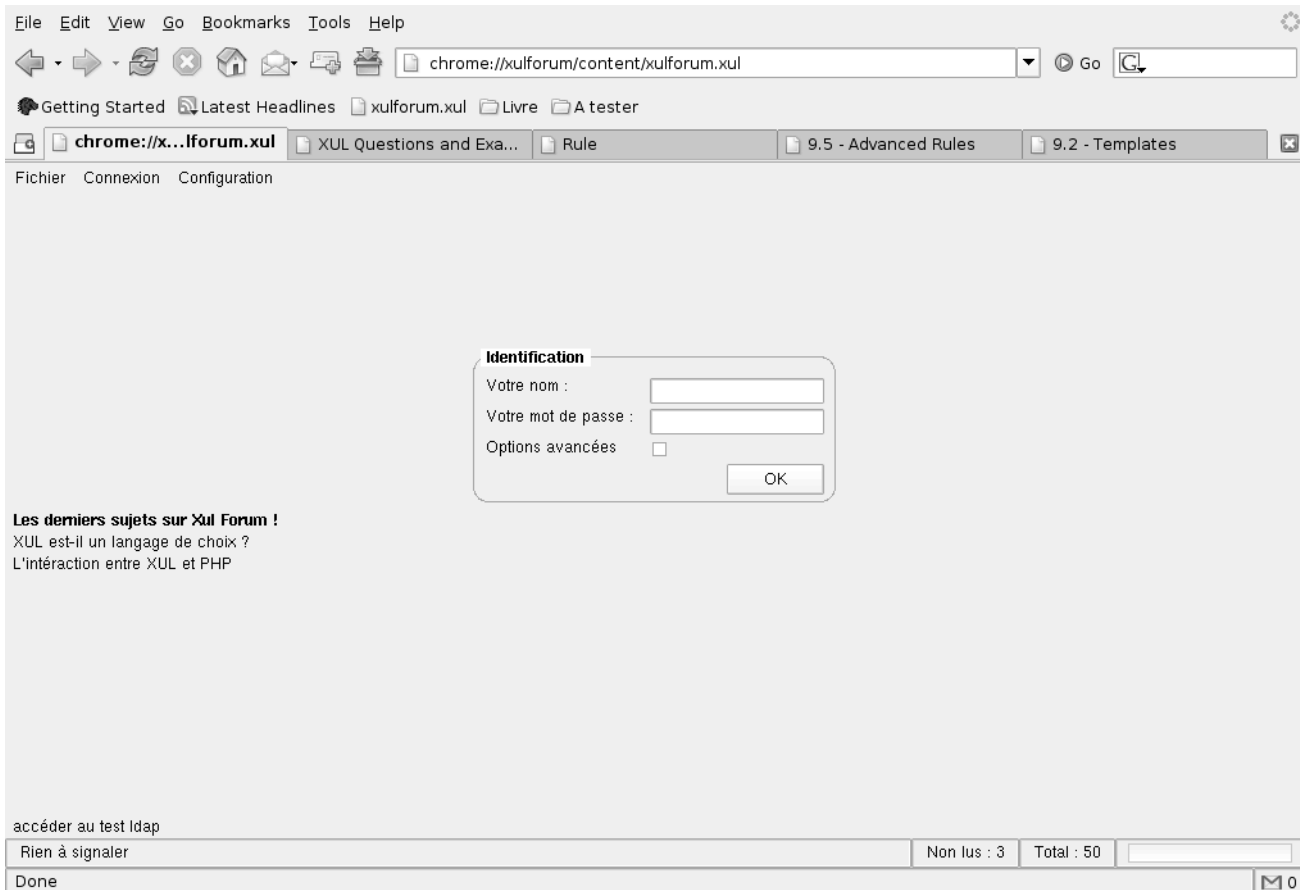


Figure 8-5 La liste des sujets créée automatiquement

Si l'on veut afficher le titre de la première réponse à ce sujet, on peut créer une règle plus complexe (qui nécessite cependant que les sujets aient un message par défaut) :

```
<!-- à ajouter dans les conditions -->
<member container="?sujet" child="?message" />
<triple subject="?message"
  predicate="http://www.xulforum.fr/rdf#titre"
  object="?titrem" />

<!-- puis dans l'action -->
<label uri="?sujet"
  value="?titre (première réponse : ?titrem )" />
```


Un modèle plus complexe

Ceci n'était qu'une introduction : nous allons directement remplir l'arbre avec les sujets et leurs réponses correspondantes grâce à une méthode similaire.

ALTERNATIVE Autres types de templates

Sur le site de référence, xulplanet.com, vous pourrez explorer bon nombre de cas typiques d'utilisation d'un modèle de templates XUL. Le plus souvent, la solution de votre problème se trouvera dans un des nombreux cas de figure offerts.

► <http://www.xulplanet.com/testcases/example-viewer.xul>

Modification côté PHP

Il nous faut d'abord modifier la création du fichier RDF, qui ne produit pour l'instant que la liste des sujets. L'ajout d'une seconde boucle, qui, pour chaque sujet, crée une liste de messages correspondants, nous permet d'obtenir le résultat suivant :

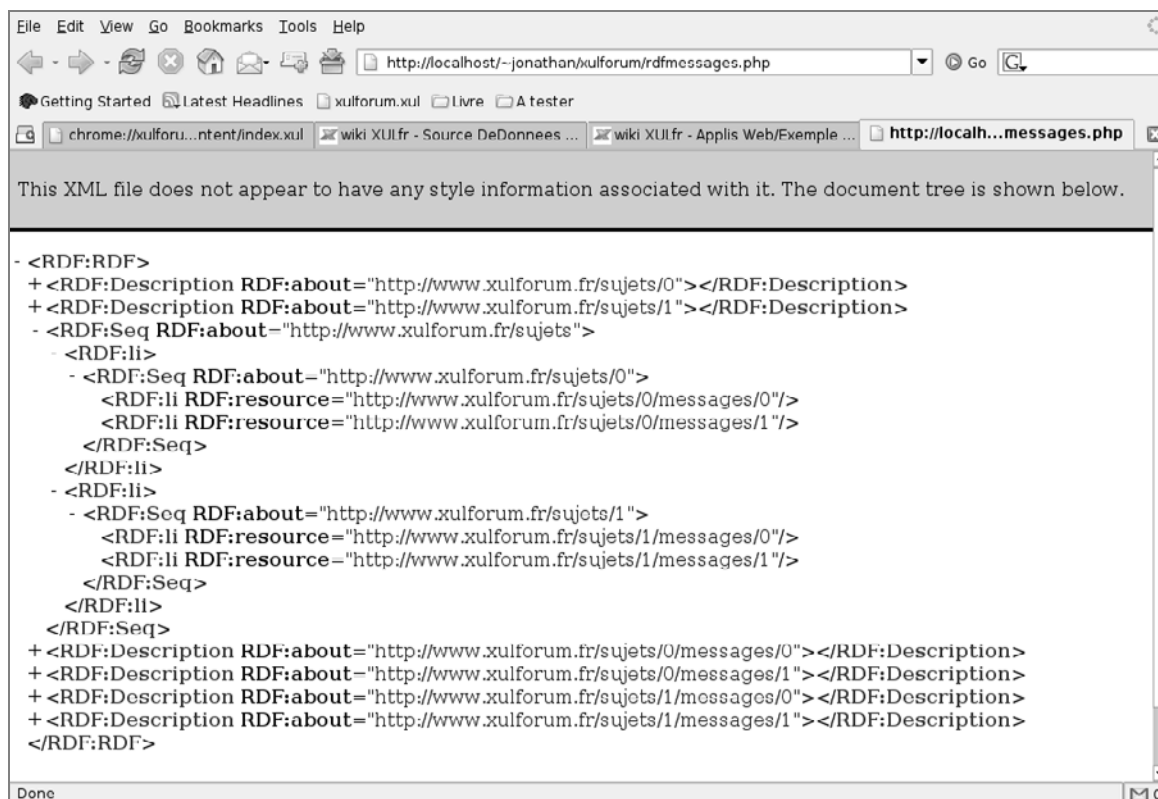


Figure 8-6 Le nouveau fichier RDF avec un niveau de profondeur supplémentaire

ASTUCE Erreurs RDF

À la suite des améliorations de la création de RDF, au lieu d'utiliser un simple `exit()` dans le bloc catch, on peut créer un « RDF de secours », c'est-à-dire un faux sujet avec une fausse première réponse, contenant l'erreur.

On peut créer entièrement le document d'erreur avec PHP, mais la tâche est assez lourde, même si cela reste faisable (voir à ce propos les lignes mises en commentaire dans `rdfmessages.php`, à récupérer sur le site du projet).

Une solution plus rapide consiste à écrire un fichier RDF de base « à la main », à le charger dynamiquement via PHP et à modifier les deux/trois attributs nécessaires.

```
<!-- le premier message du premier sujet,
avec les attributs id pour manipuler ensuite
le contenu via php -->
<RDF:Description RDF:about=
  "http://www.xulforum.org/sujets/0/
  messages/0">
  <XF:titre xml:id="titreErreur">
  </XF:titre>
  <XF:date xml:id="dateErreur"></XF:date>
  <XF:auteur>XUL Forum - serveur rdf
  </XF:auteur>
</RDF:Description>
```

Les attributs `xml:id` permettent de signifier à l'analyseur que l'attribut `id` n'est pas un attribut quelconque appartenant à un éventuel espace de nommage par défaut. En faisant appartenir cet `id` à l'espace de nommage XML, on le rend compréhensible par le DOM et on peut ainsi utiliser la méthode `getElementById()`, ce qui n'aurait pas été possible sans le `xml:` devant.

```
/* PHP : dans le bloc catch, à la place de
exit() */
$d->loadXML(file_get_contents(
  "erreurtype.xml"));
$e = $d->getElementById("titreErreur");
$e->appendChild(new
  DOMText($e->getMessage()));
$d->getElementById("dateErreur")->
  appendChild(new DOMText(date('c')));
```

Ainsi, l'on remplace juste les contenus qui changent en fonction de l'erreur et on peut communiquer efficacement avec l'utilisateur, sans surcharger le code PHP. N'oubliez pas de supprimer les erreurs grâce à l'opérateur `@` devant le nom de la fonction, sinon elles s'affichent sur la sortie standard et empêchent l'analyse du document XML.

Chaque sujet est désormais une liste : la séquence principale est une séquence de séquences. Chaque sujet contient la liste de ses messages, sous forme d'éléments `li` pointant vers des `Description`. Ceci va nous permettre d'utiliser une récursion pour l'arbre, c'est-à-dire que nous allons créer du contenu avec plusieurs niveaux de profondeur. Il y a aussi un nouvel attribut `lu` qui porte la valeur `nonlu` lorsque le sujet n'est pas lu et qui n'est pas spécifié dans le cas contraire.

Exploitation côté XUL

C'est maintenant le fichier `index-forum-overlay.xul` qui va subir quelques modifications. Notre remplissage d'exemple va être modifié au profit d'un vrai remplissage RDF :

La nouvelle version de l'arbre

```

<tree id="xf-index-arbre" flex="1" datasources=
"http://localhost/~jonathan/xulforum/rdfmessages.php"
ref="http://www.xulforum.org/sujets">
<treecols ... />
<template>
  <rule>
    <conditions>
      <content uri="?messages" />
      <member container="?messages" child="?message" />
      <triple subject="?message"
        predicate="http://www.xulforum.org/rdf#titre"
        object="?titre" />
      <triple subject="?message"
        predicate="http://www.xulforum.org/rdf#auteur"
        object="?auteur" />
      <triple subject="?message"
        predicate="http://www.xulforum.org/rdf#date"
        object="?date" />
    </conditions>
    <bindings>
      <binding subject="?message"
        predicate="http://www.xulforum.org/rdf#lu"
        object="?lu" />
    </bindings>
    <action>
      <treechildren>
        <treeitem uri="?message">
          <treerow>
            <treecell properties="?lu" label="?titre" />
            <treecell label="?date" />
            <treecell label="?auteur" />
          </treerow>
        </treeitem>
      </treechildren>
    </action>
  </rule>
</template>
</tree>

```

ATTENTION URI de base

Vous remarquerez que le `<content uri="?sujets">` de l'exemple précédent s'est mué en `<content uri="?messages">` mais la séquence de base est toujours : `http://www.xulforum.org/sujets/` !

Ce n'est en fait qu'une question de mise en forme... il est plus logique d'appeler la séquence de base `?messages` car nous considérons tous les messages du forum ! La séquence de départ reste bien sûr la séquence :

`http://www.xulforum.org/sujets.`

Voici les principaux changements par rapport au modèle précédent :

- Il y a plus d'éléments `<triple>` : titre, mais aussi auteur et date du sujet/message.
- Un nouvel élément `binding` fait son apparition : il est similaire à un `triple`, mais n'est pas forcément présent dans le fichier RDF (le serveur peut ne pas spécifier d'attribut `lu`). Si effectivement le message n'est pas lu, il permettra, grâce à sa valeur `nonlu`, de réutiliser le style vu au chapitre sur les CSS.
- L'attribut `uri="?message"` est placé sur un conteneur. Ceci est d'une importance capitale, c'est pourquoi nous allons détailler le comportement de XUL face à ce type de situation.

Il y a bien sûr plusieurs niveaux de profondeur : un pour les sujets et un autre plus en avant pour les messages propres à un sujet. Lorsque c'est un sujet qui alimente le modèle, nous sommes au premier niveau de profondeur et il ne se passe rien de particulier, les éléments `<treechildren>` `<treeitem>` sont ajoutés à l'arbre.

Mais lorsque l'on entre dans les messages en réponse à un sujet, on augmente le niveau de profondeur ! En effet, et on le voit bien sur la copie d'écran du `rdfmessages.php` modifié, la séquence principale des sujets contient les différents sujets... mais chaque sujet est lui-même une séquence ! Ce qui fait au total deux niveaux de profondeur. Et c'est ici que le comportement devient particulier. Comme l'attribut `uri` est placé sur le `treeitem`, l'intégralité du modèle est repris et ajouté comme enfant du `treeitem`.

ALTERNATIVE **Syntaxe simplifiée ?**

Il existe une syntaxe alternative pour les sources RDF ne demandant qu'un traitement basique. Elle aurait pu être utilisée pour afficher uniquement les titres des sujets, dans le premier exemple. Voici sa syntaxe :

```
<vbox datasources=
  "http://localhost/~jonathan/xulforum/rdfmessages.php"
  ref="http://www.xulforum.org/sujets">
  <label style="font-weight: bold;"
    value="Les derniers sujets sur XUL Forum !" />
  <template>
    <label uri="rdf:*"
      value="rdf:http://www.xulforum.org/rdf#titre" />
    </template>
</vbox>
```

La vision de la structure RDF est moins évidente... et de plus, cette syntaxe est à éviter dans le cas de traitements complexes. Attention donc aux solutions qui ont l'air plus simples : elles peuvent se révéler difficiles à faire évoluer !

Autrement dit, on obtient :

Simulation du modèle

```
<treechildren>
  <treeitem>
    <treerow ... />
    <!-- jusqu'ici tout correspond à un sujet généré -->
    <treechildren>
      <treeitem>
        <treerow...>
        <!-- ces trois éléments proviennent du modèle qui a
          été généré de nouveau, mais comme enfant du
          treechildren -->
```

Ceci nous permet d'obtenir du XUL conforme à la syntaxe, permettant de produire des arbres avec différents niveaux de profondeur.

La situation aurait été la même si on avait appliqué le modèle récursif à notre premier exemple : il y aurait eu des boîtes secondaires imbriquées dans la boîte principale contenant les sujets.

Cependant, l'utilisation de la source RDF n'est pas optimale : elle n'est pas rechargée assez fréquemment (les paramètres par défaut de Mozilla sont de plusieurs minutes de délai et il n'est pas possible de les modifier depuis XUL même, sauf pour des versions très récentes de Mozilla) et nous n'avons pas vu pour l'instant comment choisir dynamiquement la source de données, qui est un paramètre dépendant du forum.

Nous allons donc faire appel à JavaScript pour deux choses. D'une part pour extraire l'URL du serveur RDF depuis le fichier de configuration fourni et l'appliquer sur la source. D'autre part pour rafraîchir la source de données à intervalles réguliers et permettre ainsi un suivi rapide du forum.

Amélioration de RDF avec JavaScript

La première chose qu'il va falloir changer concerne la source de données : elle devra être choisie dynamiquement par JavaScript. Dans la page d'authentification, le script devra l'extraire du fichier de configuration et l'assigner à la boîte contenant les derniers messages du forum. Dans la page du forum, le principe sera le même : on inclura les scripts pour récupérer le fichier de configuration (transmis d'une page à l'autre via l'URL) et on assignera la bonne source de données à l'arbre.

ALTERNATIVES La vie sans RDF

L'utilisation de sources de données RDF/XML n'est qu'une méthode parmi plusieurs pour rapatrier du contenu distant. On peut citer deux grandes alternatives :

- Utiliser un fichier XML contenant uniquement les données et l'analyser côté JavaScript avec DOM, pour construire le contenu de l'arbre. Avec une fonction récursive, il serait possible de ne pas trop alourdir le code. Cette technique très en vogue et déjà mentionnée dans cet ouvrage est AJAX : Asynchronous JavaScript And XML, mais c'est surtout l'objet XMLHttpRequest qui est utilisé.

► <http://en.wikipedia.org/wiki/AJAX>

- Créer du JavaScript côté serveur et l'inclure dans XUL Forum. C'est la technique utilisée par Google dans son service Google Suggest : du JavaScript très complexe est créé côté serveur et il est directement inclus dans le client, pour un maximum d'efficacité... et de complexité ! Ceci peut se faire soit par l'intermédiaire d'une balise `<script>` pointant vers une URL distante, soit avec un objet XMLHttpRequest rapatriant le code créé par le serveur et l'incluant grâce à la fonction `eval("chaîne contenant le code js à interpréter")`. Cette méthode brutale peut néanmoins être appréciable, car le parseur RDF/XML de Mozilla a la réputation d'être lent...

► <http://www.google.com/webhp?complete=1&hl=en>

Pour choisir dynamiquement la source de données, il n'est pas besoin de la choisir dans le fichier XUL. Il faut donc utiliser pour la balise `tree` :

```
<tree id="xf-index-arbre" flex="1" datasources="rdf:null"
ref="http://www.xulforum.org/sujets">
```

Pour la balise `vbox` (fichier `xulforum.xul`), le changement est le même et ne concerne que l'attribut `datasources`.

Version synchrone

Lorsqu'on veut effectuer une opération sur un fichier distant, il y a deux possibilités. La première consiste à effectuer un appel synchrone : on appelle la fonction et JavaScript ne passe à la ligne suivante que quand l'opération sur le fichier est terminée.

Cette méthode a l'avantage d'être extrêmement simple à utiliser : il n'est pas nécessaire de mener un suivi « en parallèle » de l'opération sur le fichier ou d'attendre que l'opération soit terminée avant de passer à la suite. C'est cette méthode que nous allons d'abord utiliser.

Nous utiliserons deux fonctions : une première pour charger une nouvelle source RDF (si par exemple l'utilisateur change de fichier de configuration, il faudra charger la nouvelle source RDF) et une seconde pour mettre à jour la source (tant qu'on n'a pas changé de source, on reste sur la même et on la recharge régulièrement dans le cas où il y aurait de nouveaux messages).

POUR ALLER PLUS LOIN Écrire dans une source RDF

Si vous voulez fonder toute la partie stockage de votre application sur RDF, c'est possible. Grâce à une forte utilisation des composants XPCOM, on peut écrire dans une source RDF. Un « cache » pour XUL Forum pourrait ainsi être stocké dans un fichier RDF interne. Ceci pourrait constituer un bon – quoique très difficile – exercice...

Dans le lien ci-dessous, vous trouverez un chapitre sur RDF et dans « utilisation des composants XPCOM », un chapitre sur les fichiers.

► <http://xulplanet.com/tutorials/mozsdk/>

- 1 `chargerSourceRDF()` sera appelée après un chargement du fichier de configuration, soit dans la fonction `initialisation()`, soit après un clic sur le bouton *Recharger* dans l'écran d'identification. Elle prendra en compte le nouveau paramètre `cheminrdf` du fichier de configuration.
- 2 `mettreAJourSourceRDF()` sera appelée à intervalles de temps réguliers grâce à une fonction `setTimeout()`, comme nous l'avons vu précédemment.

Le fichier `content/javascript/rdf.js` contenant les routines de téléchargement des sources RDF

```
var gSource;
var gRdfElement; /* pour charger la source de données RDF */
function chargerSourceRDF(pId) { // exemple d'utilisation : chargerSourceRDF("xf-index-arbre");
                                // pour charger la source RDF vers l'arbre des messages
    try {
        gRdfElement = document.getElementById(pId); ❶
        var rdfService = Components.classes[
            "@mozilla.org/rdf/rdf-service;1"].getService(Components.interfaces.nsIRDFService); ❷
        gSource = rdfService.GetDataSourceBlocking("http://" + gConfig.php.hote + gConfig.php.cheminrdf); ❸
        gSource.QueryInterface(Components.interfaces.nsIRDFRemoteDataSource); ❹
        gSource.QueryInterface(Components.interfaces.nsIRDFXMLSink); ❺
        gRdfElement.database.AddDataSource(gSource); ❻
        dump("Source chargée : " + gConfig.php.hote + gConfig.php.cheminrdf + "\n");
        mettreAJourSourceRDF(); ❼
    } catch (e) {
        ajouterErreur("Erreur chargement RDF : " + e);
    }
}
```

Le maniement d'une source RDF en JavaScript est assez complexe, notamment à cause d'une grosse utilisation des composants XPCOM. Voyons comment fonctionne cet exemple :

- ❶ Pour que les mises à jour ultérieures puissent être effectuées, on stocke l'élément amené à être mis à jour dans une variable globale.
- ❷ Il nous faut un service RDF qui fournira les méthodes nécessaires au traitement RDF. On n'utilise pas la méthode `createInstance()` vue au chapitre précédent car c'est un service. Un service permet de manipuler toujours la même instance. À l'opposé, `createInstance()` renvoie un nouveau composant à chaque appel. Ainsi, lorsqu'on changera de source RDF, après un changement dans le fichier de configuration par exemple, on aura toujours la même instance du composant `rdf-service` à manipuler, ce qui évite des doublons en mémoire.

Il n'y a pas de méthode certaine pour déterminer si un composant est un service ou pas : généralement, on le devine grâce au nom.

- ③ On obtient la source grâce à l'URL fournie dans le fichier de configuration. Elle est stockée dans une variable globale pour, une fois encore, pouvoir être réutilisée dans les mises à jour ultérieures.
- ④ On demande les interfaces nécessaires (la seconde servira lors des appels asynchrones), dans une utilisation légèrement différente de ce que nous avons vu précédemment.
- ⑤ On ajoute cette source de données à l'élément concerné (et ceci permet de pallier l'élément vide `rdf:null` indiqué précédemment).
- ⑥ On demande à mettre à jour la source.

Ceci concerne donc le chargement seul de la source RDF et son assignation à l'élément. Voyons maintenant comment rafraîchir le contenu de la source RDF et construire le modèle en conséquence.

ATTENTION

Supprimer la source précédente

Dans l'écran d'identification, si par exemple vous changez de fichier de configuration à la volée, il faut penser à désassocier l'ancienne source de données, pour pouvoir ajouter la nouvelle. Ces deux lignes permettront de supprimer une ancienne source de données existante, avant d'en ajouter une nouvelle (il faut les placer juste après l'obtention du service RDF) :

```
if (gSource != null)
    gRdfElement.database.
        ➔ RemoveDataSource(gSource);
```

Les routines de mise à jour de la source

```
/* appelée régulièrement et en cas de bouton recharger */
function mettreAJourSourceRDF() {
    try {
        gSource.Refresh(true); ① //bloquant ! Appel synchrone
        gRdfElement.builder.rebuild(); ②
        dump("Source mise à jour\n");
        ajouterErreur("Source mise à jour");
    } catch (e) {
        ajouterErreur("Impossible de mettre à jour la source : "+e);
    }
}

/* appelée en tout début de script dans initialisation()
   on aura en plus des appels à mettreAJourSourceRDF
   après le changement de la source dans chargerSourceRDF */

const RDFDELAI = 60000; //mise à jour RDF toutes les minutes
function minuteurRDF() {
    mettreAJourSourceRDF();
    setTimeout(minuteurRDF, RDFDELAI);
}
```

Ici le fonctionnement est très simple : on appelle la fonction de mise à jour de la source ① et on reconstruit l'élément XUL concerné ②. L'argument `true` indique que l'appel est bloquant. La seconde fonction sert à effectuer des mises à jour régulières et sera appelée une unique fois en début de script grâce à la fonction `initialisation()`. Mais si la source est un gros fichier (s'il y a beaucoup de messages par exemple), elle peut être longue à récupérer... il faut dans ce cas-là la télécharger en arrière-plan.

► <http://www.xulplanet.com/references/xpcomref/interfaces/nsIRDFXMLSinkObserver.html>

Version asynchrone

Comme les procédures de téléchargement et d'analyse du fichier RDF se feront en arrière-plan, il faut que quelqu'un surveille ce qui se fait en parallèle du script principal.

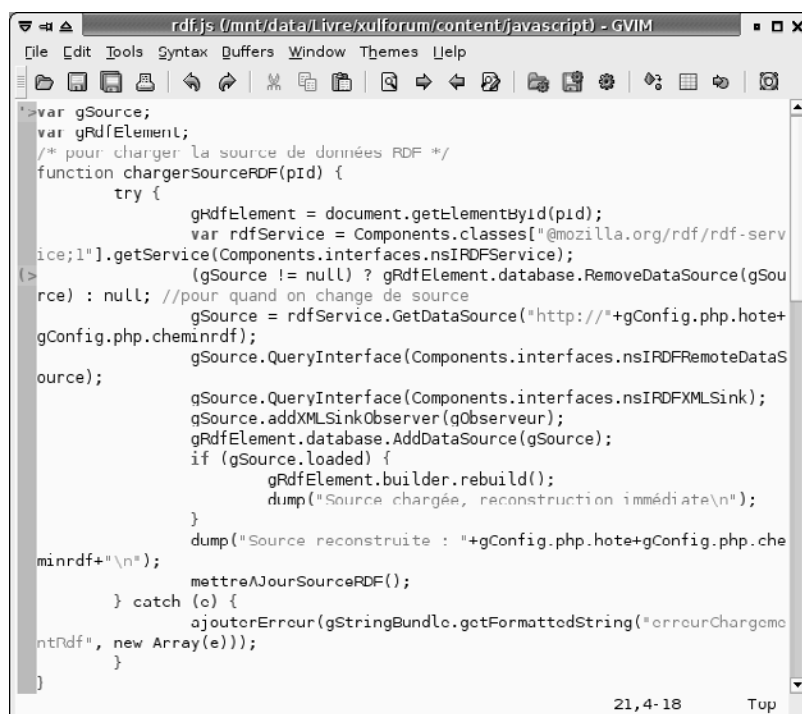
C'est le rôle de l'interface `nsIRDFXMLSinkObserver`.

Nous allons créer un objet remplissant les caractéristiques de cette interface, en JavaScript et nous allons enregistrer cet objet auprès du service RDF afin que ce dernier lui transmette des informations : mise à jour de la source, erreur éventuelle...

L'objet qui va observer l'évolution du chargement de la source

```
var gObserveur = {
  onBeginLoad : function (pSink) { dump("Chargement démarré"); },
  onInterrupt : function (pSink) { },
  onResume : function (pSink) { },
  onError : function (pSink, pStatut, pErreur)
    { ajouterErreur("Erreur au chargement ! "+pErreur); },
  onEndLoad : function (pSink) {
    ajouterErreur("Chargement terminé avec succès");
    gRdfElement.builder.rebuild();
  }
}
```

Figure 8-7
La nouvelle fonction
`chargerSourceRDF`



La fonction mettreAJourSourceRDF simplifiée

```
/* appelée régulièrement et en cas de bouton recharger */
function mettreAJourSourceRDF() {
  try {
    gSource.Refresh(false);
  } catch (e) {
    ajouterErreur("Impossible de mettre à jour la source : "+e);
  }
}
```

Il y a quelques changements mineurs par rapport à la version asynchrone. On enregistre tout d'abord `gObservateur` auprès de la source grâce à la méthode `addXMLSinkObserver`. Ensuite, lorsque l'on ajoute la source, on vérifie si elle est déjà chargée. Si c'est le cas, on reconstruit directement l'élément (et `gObservateur` n'a pas servi dans ce cas).

Si ce n'est pas le cas, la source utilise l'élément `gObservateur` pour effectuer le suivi et lorsque ce dernier voit sa méthode `onEndLoad` appelée, à la fin du chargement, il s'occupe de reconstruire l'élément.

ASTUCE Indiquer à l'utilisateur qu'une opération est en cours en arrière-plan

Nous avons dans les premiers chapitres placé un élément `progressmeter` dans la barre de statut, en bas à droite de l'écran. Il va maintenant pouvoir nous servir. Nous allons le placer en mode « indéterminé » pendant toute la durée de l'opération asynchrone.

```
document.getElementById("xf-statusbar-progres").mode =
  "undetermined"; //en début d'opération
document.getElementById("xf-statusbar-progres").mode =
  "determined";   //en fin d'opération
```



Figure 8-8 L'aspect de la barre de progression lorsque le mode est à « undetermined ».

ATTENTION Notion de source déjà chargée

Bien que nous demandions des mises à jour régulières grâce à la fonction `Refresh()`, la source est considérée comme à jour dans les cas suivants :

- lorsque que, dans l'écran d'authentification, on clique sur *Recharger* dans les options avancées et que l'on ne touche pas à l'adresse du fichier de configuration. On ne fait alors qu'ajouter une source déjà chargée précédemment ;
- lorsque l'on passe de `xul.forum.xul` à `index.xul`. La source reste en cache et ne sera rechargée que 60 secondes plus tard au prochain appel de `Refresh()`.

Dans le cas de la fonction `mettreAJourSourceRDF()`, les mises à jour sont tout le temps non bloquantes, donc l'appel aux fonctions de `gObserver` est systématique et c'est ce dernier qui se chargera de reconstruire l'élément.

ASTUCE Chaînes formatées

Les chaînes obtenues avec `gStringBundle.getString()` n'apparaissent pas dans le code pour améliorer la lisibilité. Cependant, dans le résultat final, elles sont intégrées, avec une astuce supplémentaire. En écrivant dans le fichier `.properties` :

```
erreurChargementRDFAsync=
    ➤ Erreur au chargement RDF asynchrone, erreur %s, code %s
on peut utiliser dans JavaScript :
ajouterErreur(gStringBundle.getFormattedString(
    ➤ "erreurChargementRDFAsync", new Array(pErreur,
    pStatut)));
```

En résumé...

Les fonctions RDF de XUL Forum sont désormais opérationnelles. Comme toujours, nous avons d'abord vu une solution simple et efficace, qui a été perfectionnée pour mieux répondre aux contraintes d'une vraie application : délais de réponse, latence inévitable du serveur...

- L'écran d'identification présente les derniers messages du serveur, mis à jour lorsqu'on change de fichier de configuration.
- L'écran du forum, avec sa fonction récursive, remplit parfaitement l'arbre des sujets.
- Le tout est soutenu par d'importantes fonctions de gestion RDF, en JavaScript et XPCOM.

Encore une fois, nous n'avons pas pu éviter l'utilisation de JavaScript... il va d'ailleurs faire une nouvelle apparition dans le chapitre suivant, l'intégration au cœur de Mozilla.

avaxhome

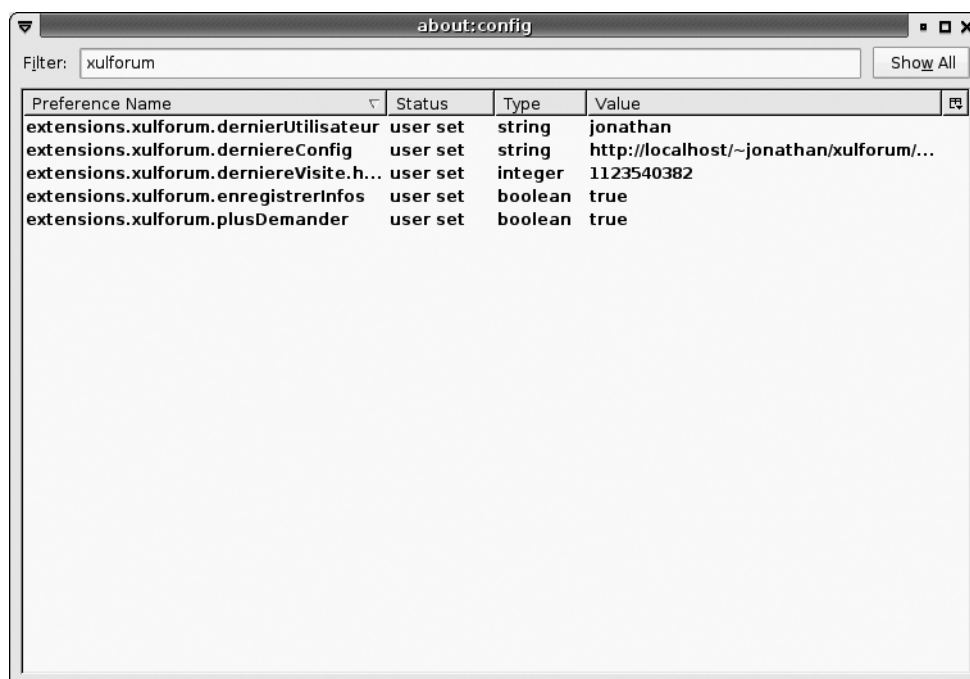
POUR ALLER PLUS LOIN Les templates XUL

Nous ne sommes bien sûr pas en mesure de décrire de manière exhaustive cet énorme pan du framework Mozilla que sont les templates. Neil Deakin, l'auteur du tutorial de XUL Planet, vous propose de découvrir leur fonctionnement en détail : à l'heure actuelle, Neil a posté 34 articles sur son blog. Au programme : requêtes SQL et templates, règles avancées, manipulation dynamique des conditions de génération... Le premier épisode est à l'adresse ci-dessous :

► <http://www.xulplanet.com/ndeakin/article/300/>

9

chapitre



The screenshot shows a web browser window with the title bar 'about:config'. Below the title bar is a search filter box containing 'xulforum' and a 'Show All' button. The main content area displays a table of preferences.

Preference Name	Status	Type	Value
extensions.xulforum.dernierUtilisateur	user set	string	jonathan
extensions.xulforum.derniereConfig	user set	string	http://localhost/~jonathan/xulforum/...
extensions.xulforum.derniereVisite.h...	user set	integer	1123540382
extensions.xulforum.enregistrerInfos	user set	boolean	true
extensions.xulforum.plusDemander	user set	boolean	true

Une intégration au cœur de Mozilla

Jusqu'à maintenant, nous avons développé XUL Forum dans le cadre du navigateur : l'accès s'y faisait par l'intermédiaire de l'adresse chrome://. Il est grand temps d'ajouter quelques améliorations : un bouton dans la barre d'outils par exemple...

SOMMAIRE

- Overlays dans le navigateur
- Enregistrement des choix de l'utilisateur dans les préférences

MOTS-CLÉS

- nsIPrefService
- contents.rdf et overlays

Dans ce court chapitre, nous présenterons deux possibilités réservées aux applications chrome. La première consiste à étendre l'interface du navigateur grâce à des overlays qui agissent directement sur les fichiers XUL de Mozilla : on pourra ainsi ajouter une entrée *XUL Forum* dans le menu *Outils* par exemple. La seconde permet d'utiliser une sorte de base de données interne : les préférences, afin de stocker des données comme le dernier nom d'utilisateur, la configuration favorite utilisée, etc.

Extension de l'interface du navigateur avec de nouveaux overlays

Modification du fichier contents.rdf

Pour que Mozilla puisse prendre en compte nos overlays, il faut les annoncer à l'avance : dire quels seront les fichiers XUL du navigateur touchés par les modifications et préciser dans quels fichiers se trouvent nos overlays. Dans les premiers chapitres, lorsque nous avons transformé les premiers fichiers XUL en une extension sommaire, nous avons mentionné de futures modifications portant sur le fichier `content/contents.rdf`. C'est ce que nous allons faire maintenant : enregistrer nos overlays dans `contents.rdf`.

ATTENTION Firefox 1.5 et Thunderbird 1.5

Avec ces versions, il n'y a plus de fichier `contents.rdf`. Reportez-vous à l'annexe A pour connaître les quelques lignes à ajouter dans le fichier `manifest`. Les modifications à `contents.rdf` ne s'appliquent donc pas si vous utilisez l'un de ces deux logiciels.

Les ajouts à `content/contents.rdf`

```
<RDF:Seq about="urn:mozilla:overlays">
  <RDF:li resource="chrome://messenger/content/messenger.xul" />
  <RDF:li resource="chrome://global/content/customizeToolbar.xul" />
  <RDF:li resource="chrome://navigator/content/navigator.xul" />
  <RDF:li resource="chrome://browser/content/browser.xul" />
</RDF:Seq>

<RDF:Seq about="chrome://messenger/content/messenger.xul">
  <RDF:li>chrome://xulforum/content/messenger-overlay.xul</RDF:li>
</RDF:Seq>

<RDF:Seq about="chrome://global/content/customizeToolbar.xul">
  <RDF:li>chrome://xulforum/content/palette-overlay.xul</RDF:li>
</RDF:Seq>

<RDF:Seq about="chrome://navigator/content/navigator.xul">
  <RDF:li>chrome://xulforum/content/mozilla-overlay.xul</RDF:li>
</RDF:Seq>

<RDF:Seq about="chrome://browser/content/browser.xul">
  <RDF:li>chrome://xulforum/content/firefox-overlay.xul</RDF:li>
</RDF:Seq>
```

Il faut d'abord décrire au travers de la séquence `urn:mozilla:overlays` quels seront les fichiers de Mozilla amenés à subir des modifications. Ce peut être les fichiers contenant les menus, les fichiers contenant la barre d'outils... ici ce sera par exemple `chrome://navigator/content/navigator.xul` (le fichier principal de la suite Mozilla). Ensuite, pour chaque fichier amené à être modifié, on décrit, toujours sous forme de séquence, les différents fichiers de XUL Forum contenant les overlays. Dans l'exemple précédent, les overlays de `chrome://navigator/content/navigator.xul` seront stockés dans l'unique fichier `chrome://xulforum/content/mozilla-overlay.xul` (les overlays auraient pu être éclatés sur plusieurs fichiers séparés). Ces modifications sont somme toute assez fixes et ne varient guère d'une extension à l'autre. La tâche intéressante consiste à chercher les fichiers à modifier...

CULTURE Cookies

Les cookies, ces petits emplacements texte permettant à un site web de stocker quelques informations sur le navigateur du client (dernière visite, nom d'utilisateur, etc.), ont longtemps constitué un mystère pour l'utilisateur « normal ». En effet, dans les options, les cookies étaient présentés de la manière suivante « Cookies are delicious delicacies » : « Les cookies sont de délicieuses friandises ». Un contributeur au projet a récemment modifié le texte en une phrase plus sérieuse expliquant la vraie fonction des cookies. Blake Ross, l'un des principaux créateurs de Firefox, eut pour commentaire « Congratulations, You've just destroyed a legend ».

L'extension Delicious Delicacies, disponible sur <http://update.mozilla.org>, permet de rétablir la phrase originale. Si vous disséquez son contenu, vous verrez que son mode d'action est exactement le même que celui que nous sommes en train d'étudier. Il existe même une variante de cette extension.

- ▶ <https://addons.update.mozilla.org/extensions/moreinfo.php?id=413>
- ▶ <https://addons.update.mozilla.org/extensions/moreinfo.php?id=227>

Où trouver les fichiers à modifier ?

Les fichiers cités plus haut paraissent pour l'instant bien obscurs. Comment trouver le fichier contenant les barres d'outils, les menus ? Où chercher au départ ? Comment connaître les id des éléments à modifier ? En se prêtant à une sorte de jeu de piste...

Il y a trois points de départ, selon que l'on agit sur Firefox, Mozilla, ou Thunderbird. Tout se concentre dans LXR, l'outil permettant de visualiser les sources de Mozilla en ligne.

- Pour la suite Mozilla, le fichier central de l'interface, celui qui décrit le navigateur, est `navigator.xul`.
- Pour Firefox, nous modifierons `browser.xul`.
- Pour Thunderbird, c'est le fichier `messenger.xul`.

navigator.xul

- ▶ <http://lxr.mozilla.org/seamoney/source/xpfe/browser/resources/content/navigator.xul>

browser.xul

- ▶ <http://lxr.mozilla.org/seamoney/source/browser/base/content/browser.xul>

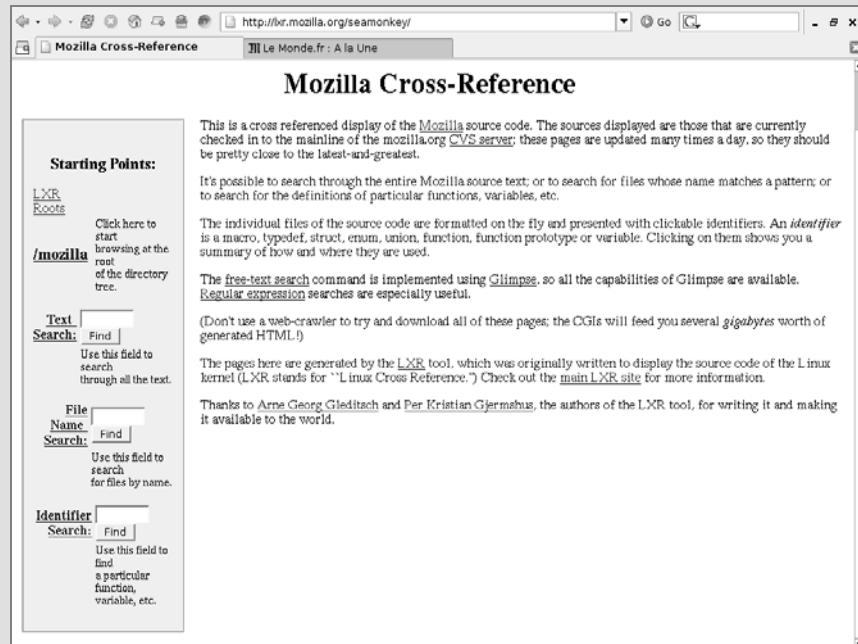
messenger.xul

- ▶ <http://lxr.mozilla.org/seamoney/source/mailnews/base/resources/content/messenger.xul>

ASTUCE Nom de fichier oublié ?

Sur <http://lxr.mozilla.org>, l'outil pour visualiser les sources de Mozilla en ligne, on retrouve trois types de recherche :

- la recherche « texte brut » : pour chercher du texte dans un fichier. Ceci vous servira pour des noms de fonctions JavaScript, pour des identifiants d'éléments XUL...
- la recherche sur un nom de fichier : vous avez oublié l'emplacement de `navigator.xul` ? C'est avec ce champ que vous le retrouverez ;
- la recherche sur un identifiant : ceci se limite au code C++ et aux fichiers de définitions d'interface IDL, que nous n'aurons pas l'occasion d'aborder dans cet ouvrage. Ceci peut servir dans des cas très extrêmes, sur des obscurs objets XPCOM qui ne seraient pas entièrement définis dans la référence offerte sur le site de XUL Planet.



ALTERNATIVE Fichiers JAR

Vous pouvez également disséquer vos archives JAR présentes dans le dossier d'installation de Mozilla et en extraire les fichiers XUL qui vous intéressent. LXR a cependant l'avantage d'être tout le temps à jour et possède des fonctions de recherche bien utiles...

Nous allons fonder notre exemple sur la suite Mozilla. Nous voulons ajouter, dans la barre de menus *Outils*, une entrée *XUL Forum* pour que l'utilisateur puisse accéder de manière conventionnelle à la future extension. Nous allons donc rechercher dans `navigator.xul` l'élément `<menubar>` qui annoncera l'arrivée du menu qui nous intéresse.

C'est la ligne 145 qui contient l'élément intéressant :

```
145 <menubar id="main-menubar" persist="collapsed"
    grippytooltiptext="&menuBar.tooltip;"/>
```

Mais cette barre de menus est vide ! En effet, `navigator.xul` contient juste le squelette de l'interface. Le gros du code est réparti sur des overlays, listés en début de fichier.

Soit en prenant un overlay au hasard (le premier), soit en effectuant une recherche sur `menubar id="main-menubar"`, on arrive à trouver le bon fichier, celui qui contient les menus : `navigatorOverlay.xul`, ligne 326.

Le premier menu est le menu *Fichier* :

```
327 <menu id="menu_File">
```

Puis le menu *Édition*, *Affichage*, *Aller*, *Favoris...* et enfin le menu *Outils* :

```
477 <menu id="tasksMenu">
478   <menupopup id="taskPopup">
```

Nous avons finalement trouvé l'identifiant de l'élément que nous allons « overlayer » : `taskPopup`.

CULTURE Les différentes branches du CVS de Mozilla

Sur <http://lxr.mozilla.org>, vous retrouverez différentes versions du code source de Mozilla (on parle de branches CVS) :

- SeaMonkey contient le code source le plus à jour, celui qui est utilisé dans les versions trunk ;
- Aviaary 1.0.1 est la branche « figée » pour les versions 1.0.x de Firefox et de Thunderbird : elle contient du code stable qui n'accepte plus que des correctifs de sécurité et des corrections de bogues ;
- Mozilla 1.7 contient aussi la branche stable du code source, pour les versions 1.7.x de Mozilla, qui n'accepte elle aussi plus que des correctifs de sécurité et des corrections de bogues ; les autres sont moins intéressantes.

De manière générale, c'est la branche SeaMonkey que vous utiliserez le plus souvent.

Gestion multiple : Firefox, Thunderbird, Mozilla

En fait, nous n'allons pas pouvoir nous contenter d'un seul overlay. L'extension devra être disponible pour Mozilla, Firefox et Thunderbird simultanément. Ce serait un vrai cauchemar de gérer trois versions concurrentes : nous allons donc faire en sorte que les fichiers de XUL Forum prennent en compte les trois applications.

ATTENTION Les différentes versions de XUL Forum

Lorsque nous arriverons au chapitre 10 concernant LDAP, nous serons obligés d'utiliser Mozilla ou Thunderbird, car Firefox ne contient pas les composants XPCOM nécessaires. En revanche, le plus grand nombre d'utilisateurs voudra tester l'extension via Firefox ou Mozilla et sans LDAP. Dernier changement, le widget XBL vu au chapitre 11 n'est valable que pour Gecko 1.8. On aura donc :

- la version finale des sources du livre : LDAP activé, compatible Thunderbird 1.5/Mozilla 1.8 ;
- la version finale modifiée, sans LDAP, compatible Firefox 1.5 et Mozilla 1.8, disponible sur le site xulforum.org, avec son forum de test.

Le plus simple est de prévoir directement l'installation dans Firefox, Thunderbird ou Mozilla, ainsi, vous serez libre ou non de suivre le chapitre 10 !

La suite Mozilla

Contrairement à Firefox, la barre d'outils de la suite Mozilla n'est pas personnalisable : on ne peut changer ni son contenu, ni l'ordre des boutons qu'elle propose. Nous n'allons donc pas imposer à l'utilisateur un bouton XUL Forum : il vaut mieux se contenter d'une entrée dans le menu *Outils*, qui sera plus discrète.

content/mozilla-overlay.xul

```
<?xml version="1.0" ?>
<?xml-stylesheet href="chrome://xulforum/skin/overlay.css"
  type="text/css" ?>
<overlay id="xf-messenger-overlay" xmlns="http://www.mozilla.org/
  ➤ keymaster/gatekeeper/there.is.only.xul">
  <menupopup id="taskPopup">
    <menuitem label="XUL Forum"
      oncommand="open('chrome://xulforum/content', null,
        ➤ 'toolbars: no, statusbar: no');" />
  </menupopup>
</overlay>
```

Le code est très simple : il se contente d'ajouter une entrée dans le menu *Outils* dont nous avons trouvé l'identifiant auparavant. Ceci fonctionne comme un overlay normal, sauf que le fichier modifié relève du cadre du navigateur et pas de notre extension.

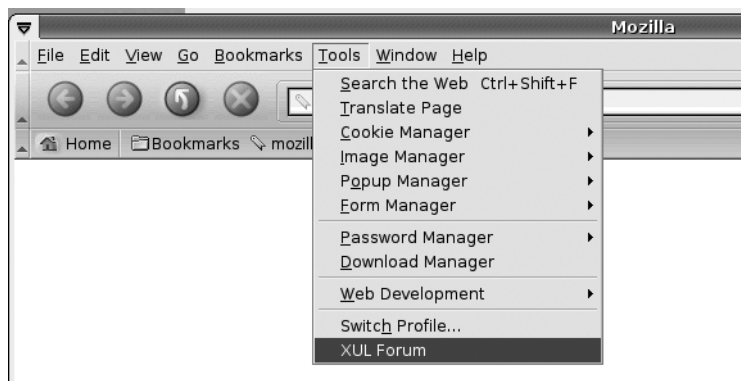


Figure 9-1

C'est bien le fichier `navigator/content/navigator.xul` sur lequel on agit : c'est celui qui possède la balise `<menubar id="main-menu">` qui est à l'origine du `<menupopup>`. Il faut travailler sur le fichier original : notre overlay ne peut pas s'appliquer à `navigatorOverlay.xul`, qui est déjà lui-même un overlay.

Le navigateur Firefox

En revanche, pour Firefox, la barre d'outils est personnalisable. Il y aura donc deux fichiers XUL à modifier : le fichier principal du navigateur (qui contiendra le bouton dans la barre d'outils) et la fenêtre permettant de choisir quels boutons ajouter ou enlever. Ici il faudra faire appel à XUL et à CSS. XUL ajoutera le bouton et CSS spécifiera des attributs essentiels : image à utiliser, changement lorsque la souris survole le bouton, etc.

CULTURE Élément toolbarpalette

Il remplace l'élément `toolbar` et permet, avec beaucoup de JavaScript et de composants XPCOM pour le glisser/déposer, de proposer une personnalisation de la barre d'outils de Firefox et Thunderbird à l'utilisateur.

Le code XUL servant à l'overlay pour Firefox

```
<?xml version="1.0" ?>
<?xml-stylesheet href="chrome://xulforum/skin/overlay.css"
  type="text/css" ?>
<overlay id="xf-messenger-overlay"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/
    there.is.only.xul">
  <toolbarpalette id="BrowserToolbarPalette">
    <toolbarbutton id="xf-toolbar-bouton"
      class="toolbarbutton-1" label="XUL Forum"
      title="XUL Forum"
      onclick="window.open('chrome://xulforum/content/',
        'XUL Forum', 'toolbars:no,statusbar:no');"
      tooltip="XUL Forum" type="button" />
  </toolbarpalette>
  <menupopup id="menu_ToolsPopup">
    <menuitem label="XUL Forum"
      oncommand="open('chrome://xulforum/content', null,
        'toolbars: no, statusbar: no');" />
  </menupopup>
</overlay>
```

L'élément `toolbarbutton` a déjà été vu lorsque nous avons construit l'interface et nous allons appliquer le même principe que dans les chapitres précédents : nous allons caractériser ce bouton grâce au CSS de la feuille de style spécifiée dans le prologue XML de l'overlay. Il possède déjà des attributs par défaut car nous l'avons fait appartenir à la classe CSS `toolbarbutton-1`, qui fixe des attributs standards pour les boutons de la barre d'outils : il ne reste plus qu'à préciser les images.

Le CSS utilisé pour l'overlay

```
/* styles pour la barre d'outils */
#xf-toolbar-bouton {
  -moz-image-region: rect(0px 24px 24px 0px);
  /* rect() a été défini au chapitre 6 */
  list-style-image: url("chrome://xulforum/skin/bouton.png");
}
```



Figure 9-2
L'image utilisée pour les boutons

```
#xf-toolbar-bouton:hover {
  -moz-image-region: rect(24px 24px 48px 0px);
}
toolbar[iconsize="small"] #xf-toolbar-bouton {
  -moz-image-region: rect(0px 16px 16px 0px);
  list-style-image:
    url("chrome://xulforum/skin/boutonpetit.png");
}
toolbar[iconsize="small"] #xf-toolbar-bouton:hover {
  -moz-image-region: rect(16px 16px 32px 0px);
}

#palette-box #xf-toolbar-bouton .toolbarbutton-text {
  display: none;
}
```

Nous utilisons deux fichiers image : un qui sert lorsque l'option *Petites icônes* est cochée et l'autre en temps normal, lorsque les icônes sont à une taille ordinaire.

Le sélecteur utilisé pour les images en petite taille, lorsqu'il est valide, écrasera celui pour les images en grande taille, car il est placé plus haut dans la cascade CSS. Ceci évite d'utiliser un sélecteur supplémentaire :

```
toolbar[iconsize="large"] #xf-toolbar-bouton.
```

Le sélecteur pour le mode *Small icons* est activé lorsque notre bouton est contenu dans une barre d'outils (élément `toolbar`) qui porte l'attribut `iconsize="small"` (spécifié dynamiquement lorsqu'on coche la case appropriée).

Enfin il y a un autre fichier à remplir, `palette-overlay.xul`. Dans ce fichier qui « overlaye » `customizeToolbar.xul`, nous plaçons encore une fois notre élément `toolbarbutton` (il apparaît de nouveau dans l'écran de personnalisation) et nous spécifions aussi la feuille de style de l'overlay, pour avoir bien sûr la bonne icône sur le bouton.

Le dernier sélecteur sert pour cet écran de personnalisation et permet d'éviter que le texte *XUL Forum* ne soit affiché deux fois. En effet, dans `customizeToolbar.xul`, le nom du bouton est déjà affiché.

Or `#xf-toolbar-bouton` est un conteneur qui contient une image et du texte en dessous (comme les boutons de la barre d'outils de XUL forum, qui affichent image et texte). Il faut donc masquer obligatoirement le texte contenu dans le bouton pour que seul l'intitulé défini par `customizeToolbar.xul` apparaisse.

Le client mail Thunderbird

Le principe est le même que pour Firefox, sauf que les attributs `id` changent : le menu pop-up porte l'`id` `taskPopup` et la palette `MailToolbarPalette`.

En ce qui concerne le fichier `palette-overlay.xul`, comme Firefox et Thunderbird possèdent tous les deux le fichier `customizeToolbar.xul`, au même emplacement, on peut placer les deux overlays relatifs aux deux éléments `toolbarpalette` dans le même fichier. Autrement dit, le fichier `palette-overlay.xul` contiendra :

Il est possible de « mélanger » des overlays

```
<toolbarpalette id="BrowserToolbarPalette">
  ...
  tout ce qui sera pris en compte
  dans le cas d'une extension Firefox
  ...
</toolbarpalette>
<toolbarpalette id="MailToolbarPalette">
  ...
  cette fois, code pour Thunderbird
  ...
</toolbarpalette>
```

Ainsi, que notre extension soit installée pour Firefox, Mozilla ou Thunderbird, les bons overlays seront à chaque fois pris en compte et si le fichier n'existe pas ou si l'`id` ne correspond à rien, le contenu sera simplement ignoré.

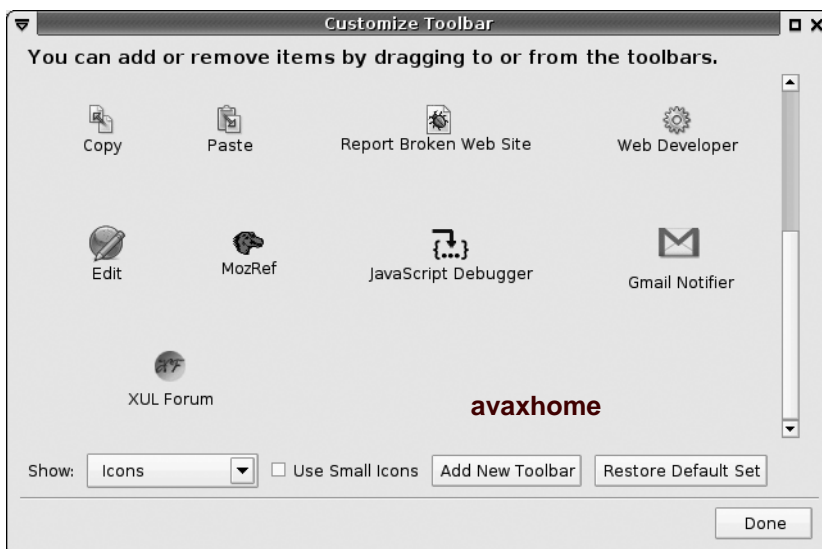


Figure 9–3

Dans le cas de Firefox, tout le contenu des fichiers `mozilla-overlay.xul` et `messenger.xul` sera ignoré, car les deux overlays portent sur des fichiers qui n'existent pas dans Firefox : `navigator.xul` et `messenger.xul`. Et la toolbarpalette portant l'id `Mail` sera elle aussi ignorée, car aucun élément de Firefox portant le même id ne pourra intégrer son contenu.

Utilisation des préférences

Une fonctionnalité agréable pour l'utilisateur consiste à se rappeler des dernières informations qu'il a entrées : dernier fichier de configuration, dernier nom d'utilisateur... ceci se fait grâce à un composant XPCOM offrant l'accès aux préférences.

Présentation

Les principales opérations sur les préférences se feront à l'aide du composant `@mozilla.org/preferences-service;1` et de son interface `nsIPrefService`. On pourra ainsi définir une branche des préférences, par exemple la branche `extensions.xulforum`. Cette convention n'est bien sûr pas obligatoire, d'autres extensions utiliseront par exemple la branche `nom_de_l_extension.nom_de_la_préférence`.

Les préférences peuvent contenir trois types de données : des entiers, des booléens (vrai ou faux) et des chaînes. Nous stockerons les paramètres suivants (tous précédés par `extensions.xulforum.`) :

- `derniereConfig`, chaîne : l'adresse du dernier fichier de configuration utilisé.
- `dernierUtilisateur`, chaîne : la dernière valeur entrée à l'identification.
- `enregistrerInfos`, booléen : pour savoir s'il faut continuer ou non d'enregistrer les dernières valeurs pour les deux paramètres précédents.
- `plusDemander`, booléen : pour déterminer si l'on continue de poser la question à l'utilisateur via une boîte modale : *Voulez-vous enregistrer ces paramètres ? Oui/Non* avec une case à cocher *Ne plus me poser la question*.
- `derniereVisite`.`http://adressedufichierdeconfiguration...`, entier : un *timestamp* représentant la dernière visite sur un forum, le forum étant identifié par le fichier de configuration qui lui correspond. Ceci permettra à PHP, lors de la création du fichier RDF, de placer l'attribut « nonlu » sur les sujets plus récents que la dernière visite.

ASTUCE Débogage des préférences

Pour voir en direct les modifications des préférences et ainsi tester vos scripts, ou pour tout simplement voir quelles sont les fonctionnalités par défaut que vous avez modifiées, l'URL `about:config` sera très utile. Elle vous permettra même de modifier certaines options de configuration avancées. Une liste assez complète se trouve sur le wiki de MozillaZine.

- http://kb.mozillazine.org/Firefox%3A_FAQs%3A_About:config_Entries

B.A.-BA Unix timestamp

Dans le monde Unix, on stocke les dates des fichiers non pas sous forme de chaîne « Samedi 30 avril 2005 », mais sous forme d'entiers, appelés timestamp (on traduirait « timbre temps »). Ces entiers représentent le nombre de secondes écoulées entre la création d'Unix (fixée par convention au premier janvier 1970) et la date représentée.

L'utilisation de ce format a plusieurs avantages :

- une place moindre : une chaîne est plus longue à stocker ;
- une interopérabilité maximale : il est très simple d'échanger des nombres entre deux langages, dans notre cas entre JavaScript côté client et PHP côté serveur et ensuite de les transformer en chaîne avec des fonctions de traitement, en PHP ou en JavaScript ;
- il n'y a aucun problème d'heure locale ou de DST (*Daylight Saving Times*).

La fonction `getTime()` de l'objet `Date` en JavaScript retourne le nombre de millisecondes, d'où la division supplémentaire par 1000.

Les fonctions XPCOM essentielles

Les fonctions de l'interface `nsIPrefService` que nous utiliserons sont au nombre de six :

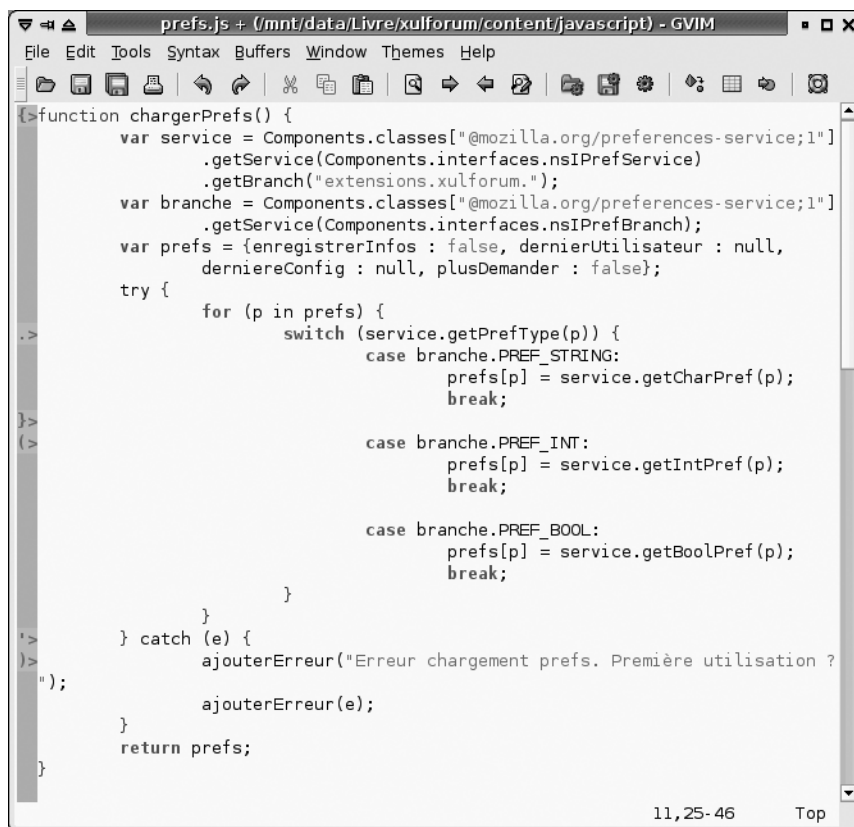
- `setBoolPref("nom de la préférence", vrai ou faux)` et `getBoolPref("nom de la préférence")`.
- `getCharPref()` et `setCharPref()` qui fonctionnent de la même manière pour les chaînes.
- `getIntPref()` et `setIntPref()`, leurs homologues pour les entiers.

Nous aurons aussi besoin d'une autre fonction : `getPrefType("nomPref")`, pour connaître le type (entier, chaîne ou booléen) d'une entrée dans la liste des préférences. Enfin, pour analyser les résultats fournis par `getPrefType()`, nous aurons besoin de trois constantes fournies par l'interface `nsIPrefBranch` : `PREF_INT`, `PREF_STRING`, `PREF_BOOL`.

Le code de XUL Forum

Pour lire les préférences, nous utilisons une fonction facile à faire évoluer. Elle se charge d'abord d'obtenir le service `nsIPrefService` pour la branche `extensions.xulforum.`, ce qui évite d'avoir à retaper le préfixe à chaque fois. Une seconde variable obtient l'interface `nsIPrefBranch`, pour les traitements ultérieurs. Ensuite, un objet contenant les valeurs par défaut des préférences est créé. On parcourt l'objet et, pour chaque clé, on regarde quel est le type de données qui lui correspond dans la table des préférences. On adapte la fonction à appeler selon le type de préférence trouvé et on modifie l'objet de départ en conséquence. Une fois toutes les variables membres de l'objet remplies avec les valeurs tirées des préférences, on peut retourner l'objet qui contient maintenant les valeurs contenues dans les préférences.

Figure 9-4
La fonction servant à
charger les préférences



Pour la fonction d'enregistrement, le principe est à peu de choses près le même : on fournit cette fois-ci un objet à la fonction et elle parcourt les variables de cet objet. Pour chaque variable, elle en détermine le type de données JavaScript – number, boolean ou string – et adapte la fonction `set...Pref()` à appeler.

Enregistrement « dynamique » des préférences

```

function enregistrerPrefs(pPrefs) {

    var service = Components.classes[
        "@mozilla.org/preferences-service;1"]
        .getService(Components.interfaces.nsIPrefService)
        .getBranch("extensions.xulforum.");

    var branche = Components.classes[
        "@mozilla.org/preferences-service;1"]
        .getService(Components.interfaces.nsIPrefBranch);

```

```

for (p in pPrefs) {
  switch (typeof pPrefs[p]) {
    case "string":
      service.setCharPref(p, pPrefs[p]);
      break;

    case "number":
      service.setIntPref(p, pPrefs[p]);
      break;

    case "boolean":
      service.setBoolPref(p, pPrefs[p]);
      break;
  }
}

```

CULTURE **typeof**

typeof est, au même titre que **instanceof**, une construction de langage. L'utilisation de parenthèses comme pour une fonction est donc facultative.

Ainsi, on pourra obtenir d'un coup les préférences de XUL Forum :

```

var o = chargerPreferences();
dump(o.dernierUtilisateur);
dump(o.derniereConfig);
...

```

Et pour les enregistrer, on crée un objet à la volée :

```

enregistrerPreferences( {
  dernierUtilisateur : "jonathan",
  derniereConfig : "http://localhost/config.xml" } );

```

Application à l'identification

Il faut d'abord commencer par charger les paramètres. Comme toujours, nous utiliserons la fonction initialisation :

Intégration des préférences au code de XUL Forum

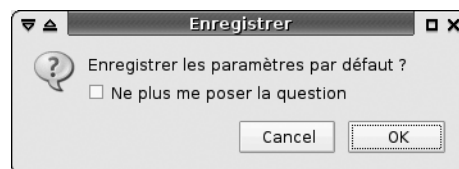
```

var gPrefs = null;
/* initialisation */
function initialisation () {
  gPrefs = chargerPrefs();
  if (gPrefs.derniereConfig != null)
    document.getElementById("xf-ident-opt-config").
      ➤ setAttribute("value", gPrefs.derniereConfig);

  if (gPrefs.dernierUtilisateur != null)
    document.getElementById("xf-ident-ident_nom").
      ➤ setAttribute("value", gPrefs.dernierUtilisateur);
  ...
}

```

Figure 9-5
La boîte de dialogue modale
qui s'affiche après l'identification



Nous utiliserons un composant XPCOM pour proposer une boîte de dialogue standard.

Figure 9-6
Le code XPCOM utilisé pour
poser la question à l'utilisateur

```
function montrerOptions() {
+-- 7 lines: dump("Affichage des options avancees\n");-----
}

var gPrefs = null;
/* initialisation */
function initialisation () {
+-- 10 lines: gPrefs = chargerPrefs();-----
}

/* pour valider le formulaire et passer à la page suivante */
function valider() {
    var r;
    var c;
    if (!gPrefs.plusDemander) {
        var prompt = Components.classes["@mozilla.org/embedcomp/prompt-
service;1"].getService(Components.interfaces.nsIPromptService);
        c = { value: false };
        r = prompt.confirmCheck(window, "Enregistrer", "Enregistrer le
s paramètres par défaut ?", "Ne plus me poser la question", c);
    }
}
```

11,17 16%

L'objet `c` représente l'état initial de la case à cocher. Il est modifié après la fermeture de la boîte pour symboliser l'action de l'utilisateur : si la case a été cochée, `c.value` vaut `true`. La variable `r` représente quant à elle la réponse de l'utilisateur : vrai s'il a cliqué `OK`, faux s'il a annulé.

On adaptera ensuite le traitement et la modification des préférences au cas par cas : faut-il demander la prochaine fois ? Faut-il enregistrer les paramètres, laisser les valeurs précédentes, ou remplir avec des valeurs par défaut ? Le traitement est long et ne présente pas d'intérêt particulier : vous pourrez le retrouver dans les sources de XUL Forum, fichier `content/javascript/identification.js`.

ALTERNATIVE L'élément <dialog>

Il existe un type de fenêtre spécial, l'élément `<dialog>` qui permet de reproduire le comportement obtenu avec le composant XPCOM. Comme XUL Forum est une extension, il est plus simple de suivre la solution XPCOM. L'élément `<dialog>` pourra être réutilisé pour une boîte *À propos de...* par exemple.

► <http://www.xulplanet.com/tutorials/xultu/dialogs.html>

Autres techniques utiles

Raccourcis clavier

Maintenant que notre application va être principalement lancée seule, c'est-à-dire non plus dans un onglet du navigateur, nous pouvons lui associer des raccourcis clavier. Auparavant, nous étions gênés car le navigateur avait lui aussi ses propres raccourcis qui pouvaient entrer en conflit avec ceux de XUL Forum (comme Ctrl + R qui recharge la page dans Firefox).

Nous pourrions ainsi rajouter deux raccourcis basiques : Echap pour quitter l'application et Ctrl + R, pour forcer le rechargement d'une source de données.

À ajouter dans un fichier XUL, pour lui associer des raccourcis clavier

```
<keyset id="xf-raccourcis">
  <key id="quitter-cle" keycode="VK_ESCAPE"
    oncommand="window.close();" />
  <key id="rafraichir-cle" key="R" modifiers="control"
    oncommand="mettreAJourSourceRDF();" />
</keyset>
```

Le premier raccourci clavier se fait à l'aide d'une touche spéciale (non alphabétique), la touche Echap, ce qui explique l'utilisation de l'attribut `keycode`. `VK` signifie *Virtual Key*, touche virtuelle en français. Il en existe beaucoup d'autres, la liste est disponible dans le fichier `nsIDOMKeyEvent.idl`.

► <http://lxr.mozilla.org/seamonkey/source/dom/public/idl/events/nsIDOMKeyEvent.idl#45>

N'oubliez pas de supprimer les préfixes `DOM_` avant d'utiliser ce code dans votre fichier XUL !

Le second raccourci se fait à l'aide d'une touche alphabétique, la touche R : c'est l'attribut le plus simple, `key`, qui est mis en jeu. L'attribut `modifiers` indique qu'en plus de la touche R, il faut presser une touche spéciale en même temps : la touche Contrôle ou Ctrl sur la plupart des claviers. Cependant, sur des Macintosh, le raccourci peut être différent (par exemple Pomme + R) et la valeur `control` ne s'applique pas. On utilisera donc `accel`, qui garantit le choix de la touche spéciale en fonction de la plate-forme et affichera Ctrl + R sous Windows ou Linux.

Nous pourrions placer les différents raccourcis dans l'overlay contenant les menus et placer deux éléments vides `<keyset id="xf-raccourcis">` dans `xulforum.xul` et `index.xul`.

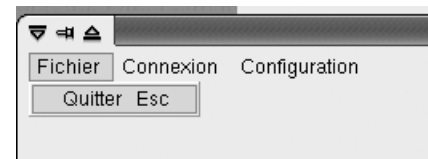
XUL permet même d'associer un raccourci clavier à une entrée de menu :

```
<menuitem key="quitter-cle"
  label="&index.menu_fichier_quitter;"
  oncommand="window.close();" />
```

Ceci fera apparaître la combinaison de touches nécessaire à droite de l'entrée de menu.

Figure 9-7

Le raccourci clavier pour quitter apparaît à droite



ALTERNATIVES Plusieurs « modifieurs »

On peut spécifier plusieurs valeurs séparées par un espace dans l'attribut `modifiers="..."`.

Par exemple :

```
modifiers="control alt"
```

obligera l'utilisateur à presser simultanément Ctrl, Alt et la touche spécifiée par `key=""` ou `keycode=""`.

Les différentes valeurs possibles sont : `control`, `alt` (option pour un Mac), `meta` (touche Commande sur un Mac), `shift`, `accel` (détaillé ci-contre).

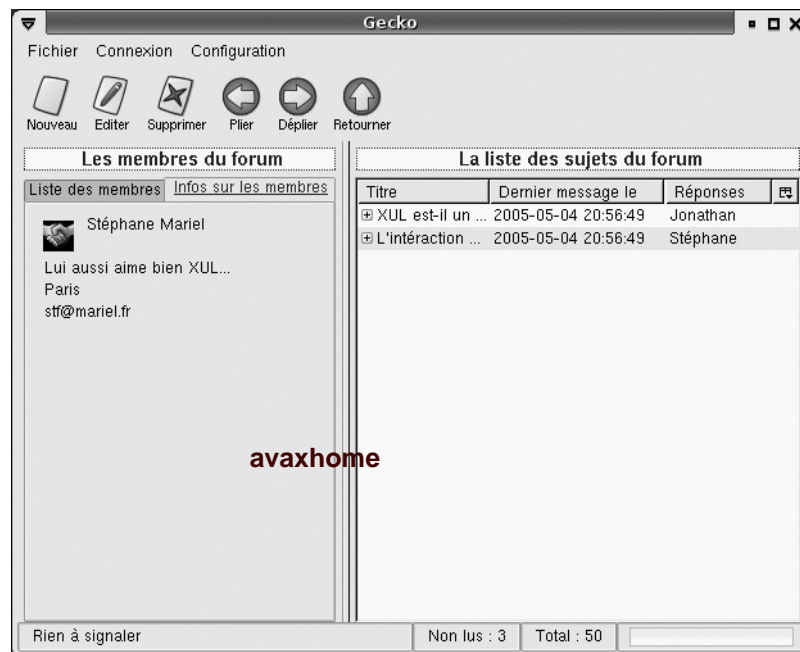
En résumé...

Dans ce chapitre nous avons totalement « fondu » notre extension dans Mozilla : échanges de données extension/navigateur via les préférences, fusion des fichiers du navigateur avec les ajouts de notre extension, raccourcis clavier...

Le chapitre suivant sera consacré à LDAP : intégration d'un service d'annuaire déjà existant dans le processus d'identification de l'utilisateur XUL Forum.

10

chapitre



JavaScript version « pro » : LDAP

Il est temps de se débarrasser une fois pour toutes de l'image de langage « simpliste » qui colle à JavaScript : dans ce chapitre, nous ferons taire les critiques en montrant qu'il est possible d'effectuer des tâches plus qu'ardues avec ce langage.

SOMMAIRE

- ▶ LDAP ? Présentation et utilisation dans le cadre de l'application
- ▶ L'identification
- ▶ La liste des connectés
- ▶ Les informations pour un connecté

MOTS-CLÉS

- ▶ Proxy XPCOM
- ▶ Listeners
- ▶ Threads asynchrones

ATTENTION Firefox et LDAP

Les composants XPCOM sont inclus uniquement dans la suite Mozilla et Thunderbird. Vous devrez donc changer de plate-forme de développement si vous choisissez de tester LDAP alors que vous utilisiez auparavant Firefox. La procédure d'enregistrement de l'extension pour Thunderbird 1.0 est exactement celle décrite au chapitre 4. Pour Thunderbird 1.5, elle est décrite en annexe A.

Outils DOM Inspector

Pour Thunderbird, le DOM Inspector est à télécharger sous forme d'extension, si vous avez besoin de l'utiliser.

- <http://www.extensionsmirror.nl/index.php?showtopic=2601>

- http://www.xulplanet.com/references/xpcomref/group_LDAP.html

Dans ce chapitre, peut-être un peu complexe, nous montrerons une utilisation inhabituelle et néanmoins utile de JavaScript : son interaction avec le service d'annuaire LDAP. LDAP nous servira de mécanisme d'identification pour l'utilisateur : dans le monde Unix, l'identification Samba, Apache ou même le *login* d'un utilisateur peuvent se faire grâce à LDAP (voir l'encadré ci-contre). L'identification XUL Forum se fera de la même manière. Ensuite, nous prolongerons l'interaction avec LDAP en tirant parti de sa deuxième fonction, certainement la plus largement utilisée : l'annuaire. Nous pourrions ainsi, après avoir ouvert l'application pour l'utilisateur enregistré, lui proposer une liste des membres, dont le contenu sera alimenté par le serveur LDAP.

Ce chapitre très technique ne doit pas être perçu comme une obligation absolue : si vous n'êtes pas particulièrement intéressé par LDAP, si vous ne vous sentez pas d'affinité immense avec les composants XPCOM, si l'installation d'un serveur LDAP ne vous tente pas nécessairement, ne vous sentez pas obligé de lire ce chapitre. LDAP ne sera d'ailleurs pas utilisé dans la version distribuée sur le site du projet, la version fonctionnelle étant uniquement fondée sur MySQL et PHP. En revanche, si vous avez déjà eu l'occasion d'utiliser ce service d'annuaire, que ce soit en entreprise ou pour votre curiosité personnelle, si vous avez envie de découvrir LDAP, si vous voulez voir un cas d'utilisation de JavaScript complexe, alors ce chapitre est fait pour vous !

B.A.-BA LDAP et Mozilla

Le lien est étroit entre LDAP et Mozilla. Le projet Mozilla propose en effet un SDK (ou Software Development Kit), qui propose une bibliothèque pour accéder à un serveur LDAP. Ce SDK, héritier des efforts portant sur le service d'annuaire, appelé Directory Server de Netscape, est utilisé par exemple dans le projet OpenOffice, ou peut servir de support pour l'extension LDAP de PHP. Pour XUL Forum, nous utiliserons la version XPCOM de ce SDK, qui n'est en fait qu'une interface xpconnect entre JavaScript et les bibliothèques en C.

- <http://www.mozilla.org/directory/>

Recherches LDAP avec JavaScript et nos propres composants XPCOM

Nous allons détailler les différentes étapes nécessaires à l'élaboration d'une recherche sur notre serveur LDAP. Toujours dans la logique d'une séparation fonctionnelle du code JavaScript, tout ce qui concerne LDAP sera stocké dans `content/javascript/ldap.js`. Le fichier fera finalement 150 lignes de code environ.

B.A.-BA LDAP... oui mais encore ?

Nous allons beaucoup parler du fameux LDAP dans ce chapitre : autant en faire une présentation immédiate.

LDAP signifie Lightweight Directory Access Protocol ou, en mauvais français, Protocole d'accès aux annuaires légers. LDAP définit en fait :

- un protocole, c'est-à-dire la manière dont doivent communiquer le client et le serveur d'annuaire, la forme sous laquelle se présentent leurs messages ;
- des modèles : comment stocker les informations et les différents types de données, comment les identifier, les ranger, etc. ;
- des API, c'est-à-dire des bibliothèques offrant des fonctions pour communiquer avec le serveur ;
- LDIF, un format pour l'échange des données entre services d'annuaires.

LDAP est conçu pour stocker de manière hiérarchique beaucoup de données, mais de petite taille.

La version actuelle du protocole est LDAP v3.

Les entrées LDAP suivent des modèles : il existe des modèles pour une personne (son nom, sa description, sa photo...), pour un contact (ajout d'un champ mail) pour un utilisateur de système Unix (il y aura alors un mot de passe), etc. Ceux-ci sont stockées dans le champ `objectClass` d'une entrée. Nos entrées d'utilisateurs XUL Forum font partie des classes `top`, `inetOrgPerson` et `person`.

Vous pouvez administrer votre serveur LDAP en ligne de commande ou en utilisant des outils graphiques comme phpLDAP-admin. Si vous désirez approfondir vos connaissances, consultez la page de Wikipedia :

► <http://en.wikipedia.org/wiki/LDAP>

ou cet article très complet de Laurent Mirtain :

► <http://www-sop.inria.fr/semir/personnel/Laurent.Mirtain/ldap-livre.html>

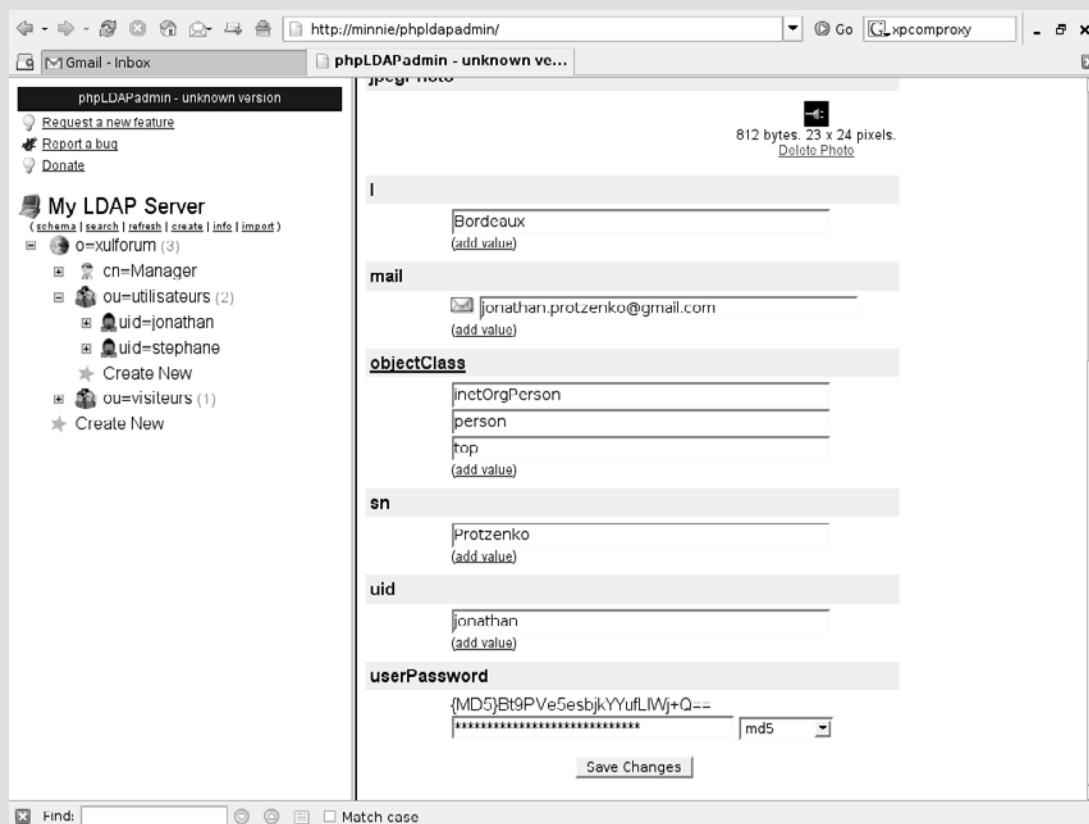


Figure 10-1 Consultation d'une entrée à l'aide du logiciel d'administration phpLDAPadmin

POUR ALLER PLUS LOIN

Les fichiers de configuration

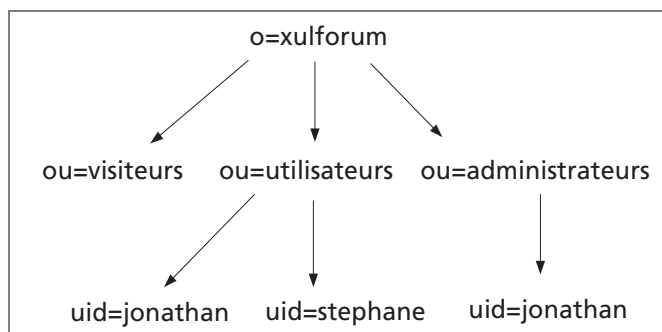
Le serveur retenu lors du développement est OpenLDAP, une implémentation libre et reconnue de ce protocole. Vous pourrez retrouver les détails de sa mise en œuvre (fichiers LDIF de base, guides utilisés, fichiers de configuration `slapd.conf`) sur le site web du projet.

Pour les composants XPCOM et les interfaces auxquelles il sera fait référence dans la suite du chapitre, la référence se trouve comme toujours sur le site de XUL Planet.

La structure LDAP de XUL Forum : le DIT

Nous allons tout d'abord présenter la structure du serveur LDAP utilisé pour XUL Forum. Il est structuré de manière hiérarchique : au sommet se trouve le nœud racine : ce sera l'organisation XUL Forum, notée `o=xulforum`. Ce type de notation s'appelle un DN (pour Distinguished Name) et permet d'identifier un élément dans un arbre hiérarchique LDAP. On définira ensuite plusieurs nœuds fils : `ou=utilisateurs` (ou signifiant *organizational unit*), `ou=visiteurs`, `ou=administrateurs`... Et dans chacun de ces groupes, on définira des personnes : `uid=jonathan` (`uid` comme *user id*) par exemple. Comme Jonathan pourra être intégré à la fois dans les utilisateurs et les administrateurs, on précisera `uid=jonathan,ou=administrateurs,o=xulforum` pour être certain de manipuler la bonne entrée. Le schéma ci-dessous récapitule cette structure : il s'appelle le Directory Information Tree ou DIT.

Figure 10-2
Le DIT pour le serveur
LDAP de XUL Forum

**ALTERNATIVES Base de données classique MySQL (ou autre)**

Il aurait bien sûr été possible d'utiliser un mécanisme d'identification à l'aide du célèbre couple PHP/MySQL. On peut en effet reprocher à LDAP une spécificité trop grande : en dehors d'un milieu intranet d'entreprise, qui utilise LDAP pour ses besoins personnels, ou même, qui utilise LDAP dans son forum sur le Net ?

La solution PHP/MySQL a été retenue pour la version « fonctionnelle » de XUL Forum, testable sur le site du projet. Elle n'inclut pas les composants LDAP et se fonde sur des appels à des services web, des `XMLHttpRequest` pour pallier le manque d'utilisations concrètes de LDAP. Le serveur PHP compare les noms d'utilisateurs et les mots de passe fournis avec les entrées présentes dans la base de données MySQL et renvoie vrai ou faux si l'identification a réussi. De même, la liste des membres sera remplie grâce à une source RDF gérée là encore par PHP.

La succession des différentes fonctions

Nous allons présenter l'enchaînement logique des différentes fonctions du fichier `ldap.js` de XUL Forum, pour ensuite nous intéresser au contenu de chacune d'entre elles. Cette approche permettra d'éclaircir progressivement une procédure au premier abord complexe.

Structure globale du fichier `content/javascript/ldap.js`

```
var gLdapFonctions = { };

/* le listener pour l'objet nsILDAPConnection*/
function globalListener () { }

/* celui pour l'objet nsILDAPOperation*/
function LDAPListener () { }

/* la fonction pour servir les files d'attente pour les
listeners */
function obtenirProxy(pObjet, pInterface) { }

/* pour servir un nouveau nsILDAPOperation tout prêt ! */
function obtenirOp() { }

/* la recherche */
function chercher() { }

/* initialisation des variables */
function initialiserLDAP(pDn, pHost) { }
```

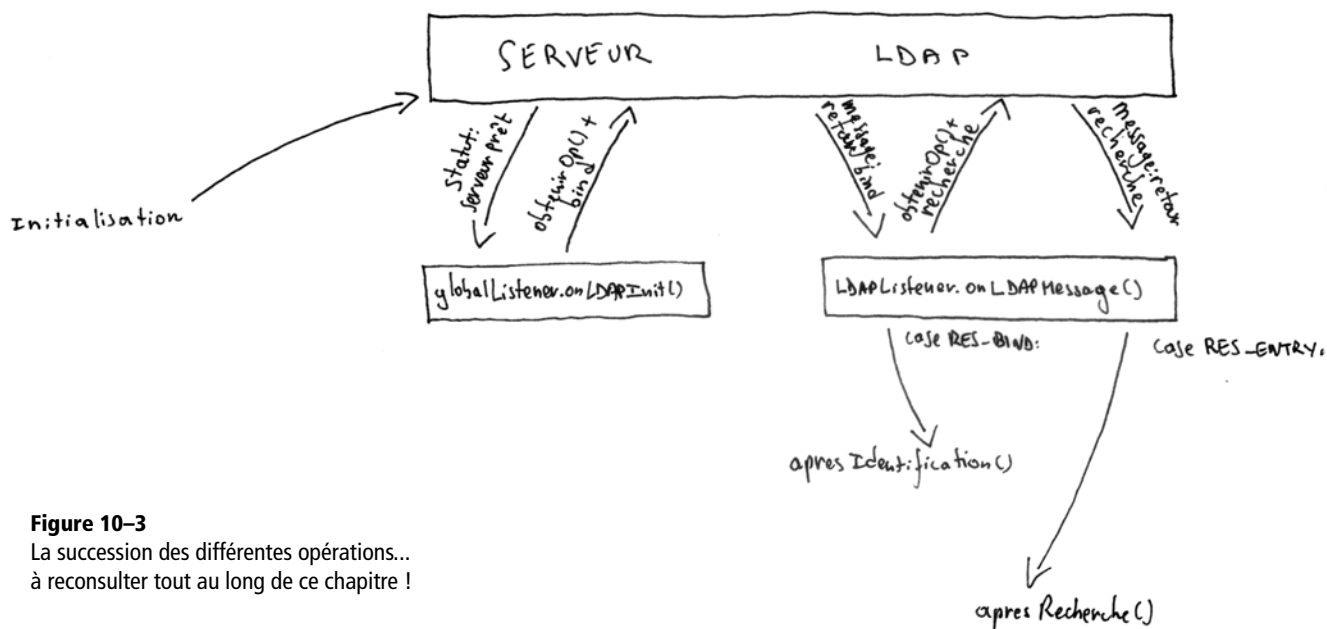
Tout commence avec la fonction `initialiserLDAP`. Elle prend comme argument le DN qui servira de base pour les recherches et l'hôte auquel le script doit se connecter. Cette fonction se charge donc d'initialiser toutes les caractéristiques de la connexion au serveur : version du protocole, tri éventuel sur les résultats renvoyés, étendue de la recherche (uniquement le DN de base choisi, ou bien ses sous-éléments par exemple).

Une fois les paramètres choisis correctement, nous lançons la connexion proprement dite. C'est ici que le script adopte un comportement particulier, différent des méthodes procédurales que nous avons vues auparavant. L'opération consistant à rapatrier les résultats d'une opération LDAP a une durée indéterminée et on ne peut pas se permettre d'attendre qu'elle soit achevée pour continuer le script. Si l'on attendait la fin de la connexion au serveur (qui peut être très longue, suivant les aléas du réseau), tout le reste du JavaScript serait figé, notamment l'affichage. Il faut donc effectuer une opération asynchrone : les opérations continueront lorsque les résultats seront arrivés et, dans le temps d'attente, le script n'est pas bloqué.

POUR ALLER PLUS LOIN **Code dans LXR**

Nous l'avons souvent souligné, il est crucial de trouver un exemple de code pour comprendre comment fonctionnent les composants XPCOM. Ici, c'est dans LXR que l'on trouve cet exemple. Le fichier indiqué sert à rapatrier des certificats de sécurité pour le gestionnaire de contacts.

- ▶ <http://lxr.mozilla.org/seamMonkey/source/mailnews/extensions/smime/resources/content/certFetchingStatus.js>

**Figure 10-3**

La succession des différentes opérations...
à reconsulter tout au long de ce chapitre !

Nous lançons donc la connexion LDAP et, opération cruciale, nous enregistrons un *listener*. Ce listener est un objet dont certaines méthodes seront appelées à un moment judicieux. Ainsi, lorsque la connexion LDAP aura fini son initialisation, une méthode du premier listener, `gLdapListener`, sera appelée. Cette méthode agira en conséquence, d'abord en demandant l'accès à une opération LDAP (`obtenirOp()`) et en lançant ensuite cette opération : c'est l'identification.

C'est alors le second listener qui est appelé, lorsque les résultats de l'identification arrivent. Ce listener qui traite les retours des opérations LDAP analyse le message qui lui est fourni : si c'est le résultat d'une identification qui arrive, il lance la recherche (c'est la fonction `chercher()`), après avoir obtenu une autre opération LDAP.

La recherche est transmise au serveur LDAP, qui l'effectue et renvoie les résultats. Une fois ces résultats arrivés, c'est toujours le second listener qui est mis en œuvre : il analyse le ou les messages qui lui sont fournis, en décomposant les différents constituants du ou des DN trouvés lors de la recherche.

Une fonction n'a pas été abordée : c'est `obtenirProxy()`. Elle permet en fait d'appeler un composant XPCOM spécifique qui crée une file d'attente vers les listeners. Ainsi, quand les résultats de la recherche arrivent, ils sont mis dans une file d'attente pour être traités les uns à la suite des autres.

Il faut bien comprendre que toutes ces opérations s'effectuent « en parallèle » du script principal : les retours peuvent ainsi arriver plusieurs secondes après le clic demandant l'identification. Il faut donc penser l'identification différemment : c'est le script `ldap` qui appellera une fonction `passerPageSuivante()`, uniquement lorsque les résultats de l'identification seront arrivés. De même, une fonction `remplirListeMembres(tableau des infos du membre)` sera appelée lorsqu'un résultat de la recherche arrivera.

Pour ne pas dépendre d'un nom de fonction particulier, les fonctions à appeler seront stockées sous forme de pointeurs de fonctions dans l'objet `gLdapFonctions`, car elles peuvent changer selon que l'on est sur `identification.js` ou `forum.js`.

ASTUCE Les pointeurs de fonction

Cette terminologie, héritée du langage C, désigne en fait une astuce très simple. En voici un exemple :

```
var f; //le "pointeur de fonction"
f = alert;
f("Info de debug"); //info voyante
f = dump;
f("Info de debug"); //info discrète
```

Ceci permet d'éliminer la spécificité du traitement LDAP. Dans notre recherche, la fonction à appeler lorsque les résultats de la recherche viennent d'arriver est susceptible de varier.

L'initialisation

Nous pouvons maintenant nous intéresser à la procédure d'initialisation.

Initialisation de la partie LDAP de XUL Forum

```
/* initialisation des variables globales */
var gLdapConn = Components.classes[
    "@mozilla.org/network/ldap-connection;1"].createInstance().
    ➔ QueryInterface(Components.interfaces.nsILDAPConnection); ❶
var gLdapURL = Components.classes[
    "@mozilla.org/network/ldap-url;1"].createInstance().
    ➔ QueryInterface(Components.interfaces.nsILDAPURL); ❷
var gLdapOp; ❸

function initialiserLDAP(pDn, pHost) {
    dump("Début initialisation gLdapURL\n");
    /* paramétrage pour l'objet nsILDAPURL */
    gLdapURL.dn = pDn;
    gLdapURL.host = pHost;
    gLdapURL.scope = gLdapURL.SCOPE_SUBTREE;
    gLdapURL.addAttribute("cn");
```

POUR ALLER PLUS LOIN

Similitudes entre JavaScript et C

Si vous connaissez le C, vous remarquerez que dans ce chapitre, beaucoup de code LDAP ressemble à du code C. `gLdapURL` est une structure plutôt qu'un objet JavaScript. Les paramètres `count` que nous verrons plus loin sont des pointeurs vers des entiers plutôt que des variables JavaScript. De toute évidence, ils permettent de faire des boucles `for` du C et non pas des boucles `for in` du JavaScript. En fait, cette extension LDAP est quasiment une version JavaScript de ce qui se fait en C, ce qui explique sa complexité par rapport aux traitements auxquels nous sommes habitués.

ALTERNATIVE Il n'y a pas que XULPlanet.com dans la vie !

Un autre site propose de la documentation créée automatiquement à partir du code source de Mozilla. On y trouve notamment des diagrammes montrant les relations entre les différentes interfaces ainsi que les commentaires Doxygen des interfaces. Pour LDAP, l'adresse est la suivante :

► <http://unstable.elemental.com/mozilla/build/latest/mozilla/directory/dox/hierarchy.html>

```
gLdapURL.addAttribute("uid");
gLdapURL.filter = "(objectClass=*)"; //tous les résultats
dump("Nous utilisons les paramètres fournis : "+pDn+
    " et "+pHost+"\n");

/* initialisation du nsILDAPConnection */
gLdapConn.init(gLdapURL.asciiHost, gLdapURL.port,
    gLdapURL.options, gLdapURL.dn,
    obtenirProxy(new globalListener,
        Components.interfaces.nsILDAPMessageListener),
    null, gLdapConn.VERSION3);
}
```

L'objet central, le composant XPCOM sur lequel repose toute notre démarche est [@mozilla.org/ldap-connection;1](http://mozilla.org/ldap-connection;1) et son interface `nsILDAPConnection`. On commence par l'instancier ❶, puis on instancie aussi un composant implémentant l'interface `nsILDAPURL` ❷. Ceci ne se fait qu'une fois, lorsque le fichier LDAP est inclus depuis le fichier principal (`forum.js` ou `identification.js`). On prépare aussi une variable globale correspondant à une opération « vide » ❸, obtenue ultérieurement grâce à `obtenirOp()` et pouvant servir pour une identification ou une recherche par exemple. `gLdapOp` est en quelque sorte un objet générique pouvant être utilisé à différentes fins.

Ensuite, on remplit les différents paramètres de `gLdapURL`, paramètres susceptibles de changer en fonction des cas : hôte, DN de base, etc. C'est pour cela que ce paramétrage est placé dans la fonction `initialiserLDAP`. On ajoute aussi les caractéristiques recherchées dans les entrées (CN et uid notamment).

Une fois que l'URL est entièrement paramétrée, on initialise la connexion, avec les paramètres choisis précédemment. Tous parlent d'eux-mêmes, sauf la fonction `obtenirProxy()` : c'est elle que nous allons détailler dans la partie suivante et que nous avons déjà mentionnée : elle crée une file d'attente vers un composant XPCOM que nous avons instancié.

Nos propres composants XPCOM en JavaScript : listeners

Les composants

L'un des paramètres qu'accepte la fonction `init` de l'interface `nsILDAPConnection` est un `nsILDAPMessageListener` `messageListener`. On pourrait penser au premier abord que c'est un autre composant XPCOM qu'il faut instancier, un composant implémentant cette interface.

ATTENTION Débogage, LDAP et erreurs...

Les composants LDAP XPCOM sont un petit peu pointilleux quant aux paramètres qu'on leur fournit. Souvent, des bogues se produisent de manière inexpliquée et se traduisent uniquement par une exception obscure qui n'apporte que peu d'informations quant à leur origine.

Le code proposé ici a cependant été testé et fonctionne. L'idéal est en fait de d'abord mettre au point le code récupérable en ligne, puis de le modifier ligne par ligne et, à chaque fois, de vérifier que tout fonctionne encore.

Une autre approche est de vérifier d'abord le serveur LDAP (via PHP par exemple, qui propose une interface de manipulation plus simple, ou les outils en ligne de commande s'ils ne vous font pas peur), car la configuration du serveur peut se révéler parfois difficile, puis de s'attaquer à la version JavaScript. Ce langage tiers peut vous aider à trouver d'éventuelles erreurs de fonctionnement provenant du serveur, plus difficiles à identifier en JavaScript !... Prudence donc !

Mais comment personnaliser sa réaction aux messages qui lui sont transmis ? D'ailleurs il n'existe pas de composant `@mozilla.org/ldap-message-listener;1`, mais juste des composants de traitement (pour le carnet d'adresses par exemple), qui implémentent cette interface. C'est donc un composant XPCOM que nous allons créer, qui implémentera trois méthodes : l'une, nécessaire à tout composant, est bien sûr `QueryInterface`. Les deux autres sont `onLDAPInit` et `onLDAPMessage` et sont requises par `nsILDAPMessageListener`.

Nos composants XPCOM implémentant `nsILDAPMessageListener`

```
/* le listener pour l'objet nsILDAPConnection*/
function globalListener () { }
globalListener.prototype.QueryInterface = function(iid) {
    if (iid.equals(Components.interfaces.nsISupports) ||
        iid.equals(Components.interfaces.nsILDAPMessageListener))
        return this; ❶
    else
        Components.returnCode =
            Components.results.NS_ERROR_NO_INTERFACE;
    return null; ❷
}

globalListener.prototype.onLDAPInit = function(pConn, pStatus)
{
    ... traitement ... ❸
}

globalListener.prototype.onLDAPMessage = function(pMsg) { }
// non utilisé ❹
```



```

/* celui pour l'objet nsILDAPOperation*/
function LDAPListener () { }
LDAPListener.prototype.QueryInterface = function(iid) {
  if (iid.equals(Components.interfaces.nsISupports) ||
      iid.equals(Components.interfaces.nsILDAPMessageListener))
    return this;
  else
    Components.returnCode =
      Components.results.NS_ERROR_NO_INTERFACE;
  return null;
}

LDAPListener.prototype.onLDAPInit = function(a, b) { }
// non utilisé ⑤

LDAPListener.prototype.onLDAPMessage = function (pMsg) {
  switch (pMsg.type) {
    ... traitement ... ⑥
  }
}

```

On crée deux composants séparés : le premier sert pour la connexion et le second pour les messages renvoyés par le serveur.

`globalListener` doit d'abord répondre à la fonction `QueryInstance()`. Il implémente l'interface de base `nsISupports`, que chaque composant XPCOM doit implémenter et l'interface `nsILDAPMessageListener`. Ainsi, si c'est l'une des deux interfaces qui est demandée lors de l'instanciation, on renvoie un nouvel objet ①. Si il y a méprise et que Mozilla demande une interface que n'implémente pas notre composant, on signale ceci par une erreur ②. La méthode suivante est `onLDAPInit` ③, qui est appelée une fois que la connexion est correctement initialisée ; nous verrons son contenu au paragraphe suivant.

POUR ALLER PLUS LOIN Enregistrer le composant

Les objets que nous créons remplissent les caractéristiques d'un composant XPCOM implémentant l'interface `nsILDAPMessageListener`, mais ne sont pas enregistrés pour autant auprès du navigateur. Toutefois, il serait possible d'enregistrer nos propres composants pour ensuite instancier « `@xulforum.org/ldap-listener-perso;1` ». Le lien ci-dessous vous permettra d'approfondir cette technique, avec notamment la création de l'interface en utilisant IDL.

C'est la technique utilisée dans l'extension Gmail Notifier par exemple (pour vous signaler de nouveaux courriers électroniques sur votre compte Gmail). Un composant est instancié (il vérifie régulièrement l'arrivée de nouveaux courriers), mais une seule fois : un seul composant est en mémoire. Ensuite, pour chaque fenêtre, le script demande au composant l'état actuel de la boîte. Ceci permet de centraliser les requêtes sur le serveur et de synchroniser l'affichage des différentes fenêtres.

► http://www-igm.univ-mlv.fr/~dr/XPOSE2004/xpcom/composant_js.php

La méthode `onLDAPMessage` doit être implémentée, elle est imposée par l'interface, mais elle n'est pas utilisée ④.

Le second composant, `LDAPListener`, fonctionne de la même manière. La seule différence avec le premier composant est sa méthode `onLDAPInit` ⑤ qui n'est pas utilisée (car la connexion est déjà initialisée) et sa méthode `onLDAPMessage` ⑥ qui, au contraire, est très utilisée : c'est ici que se fera tout le traitement des messages en provenance du serveur.

Création d'une file d'attente

Les proxy XPCOM pour un objet

```
/* la fonction pour servir les files d'attente pour les
   listeners */
function obtenirProxy(pObjet, pInterface) {

    var eqService = Components.classes[
        "@mozilla.org/event-queue-service;1"].getService(
        Components.interfaces.nsIEventQueueService);

    var uiQueue = eqService.getSpecialEventQueue(
        Components.interfaces.nsIEventQueueService.
            UI_THREAD_EVENT_QUEUE); ①

    var xpProxy = Components.classes[
        "@mozilla.org/xpcomproxy;1"].getService(
        Components.interfaces.nsIProxyObjectManager);

    return xpProxy.getProxyForObject(uiQueue, pInterface,
        pObjet, 5); ② //5 = 4 & 1

    /*
    http://lxr.mozilla.org/seamonkey/source/xpcom/proxy/public/
    nsProxyEvent.h
    #define PROXY_SYNC    0x0001
    // acts just like a function call.
    #define PROXY_ASYNC    0x0002
    // fire and forget. This will return immediately and you
    // will lose all return information.
    #define PROXY_ALWAYS 0x0004
    // ignore check to see if the eventQ is on the same thread
    // as the caller, and alway return a proxied object.
    */
}
```

Cette fonction sert d'abord à obtenir ① un type de file d'attente spécial, puis à instancier un proxy XPCOM ②, envoyant vers la file d'attente obtenue plus haut les objets (paramètre 3) implémentant l'interface passée en deuxième paramètre. Le dernier paramètre trouve son explication en cherchant directement dans les fichiers headers.

Quel est l'intérêt d'utiliser un proxy XPCOM pour ce composant ? Il faut en fait savoir que la plupart des composants Mozilla n'ont pas été conçus de manière *thread-safe*. Autrement dit, ils ont été pensés comme si un seul processus devait y accéder au même moment. Si l'on n'utilisait pas ce *event queue*, si deux messages arrivaient au même moment (problèmes de réseau), on aurait de la mémoire allouée/rendue par des threads différents, d'où des problèmes évidents. Il faut donc mettre une file d'attente, pour éviter que des messages n'arrivent en même temps.

Obtention d'une opération

Sous ce titre un peu abstrait se cache en fait une réalité des composants LDAP : avant de demander quoi que ce soit au serveur, il faut obtenir une opération, comme nous l'avons mentionné plus haut.

Il faut initialiser les opérations LDAP avant de « parler » au serveur

```
function obtenirOp() {
    gLdapOp = Components.classes[
        "@mozilla.org/network/ldap-operation;1"].createInstance().
        ➔ QueryInterface(Components.interfaces.nsILDAPOperation);
    gLdapOp.init(gLdapConn, obtenirProxy(new LDAPListener,
        Components.interfaces.nsILDAPMessageListener), null);
}
```

Le code est peut-être plus parlant que pour les proxies XPCOM : on instancie l'opération et on l'initialise en lui fournissant dans l'ordre :

- la connexion à laquelle elle se rapporte ;
- l'objet implémentant `nsILDAPMessageListener`, qui servira à traiter les messages résultant de cette opération ;
- un éventuel objet pouvant servir à la fermeture de l'opération : nous n'en aurons pas l'utilité.

Identification avec un simple bind

Maintenant que la structure de base est en place, nous pouvons enrichir ce squelette LDAP avec quelques traitements. Le premier sera l'identification.

```
globalListener.prototype.onLDAPInit = function(pConn, pStatus)
{
    dump("\n    ** Bound as "+pConn.bindName+"\n");
    obtenirOp();
    gLdapOp.simpleBind(gLdapFonctions.pass);
}
```

Le premier listener, se rapportant à la connexion LDAP, verra sa méthode `onLDAPInit` appelée après l'initialisation. Il réagit en affichant le DN de base fourni (par exemple `uid=jonathan,ou=utilisateurs,o=xulforum`) sur la console et en demandant une identification pour ce même DN. Vous remarquerez que le mot de passe est stocké dans `gLdapFonctions`, car il est fourni depuis « l'extérieur » de `ldap.js` : depuis `xulforum.xul` ou `index.xul`.

C'est alors le second listener qui entre en jeu, puisque les résultats d'une opération (comme l'identification) sont traités par `LDAPListener`.

Réactions lorsque la réponse de l'identification arrive

```
LDAPListener.prototype.onLDAPMessage = function (pMsg) {
  switch (pMsg.type) {
    case Components.interfaces.nsILDAPMessage.RES_BIND :
      dump("\n---BIND\n");
      var r = pMsg.errorCode ==
        Components.interfaces.nsILDAPErrors.SUCCESS;
      dump("  Succes : "+r+"\n");

      if (gLdapFonctions.apresIdentification != null && r)
        gLdapFonctions.apresIdentification();
      else if (gLdapFonctions.apresIdentification != null && !r)
        ajouterErreur(gStringBundle.getString("LDAPInvalide"));
      chercher();
      break;
  }
}
```

On envisage ici le traitement qui arrive lorsque le message est de type `RES_BIND` c'est-à-dire un retour d'identification. On vérifie d'abord le succès de l'opération : identification réussie ou non. Ensuite, si le script principal a défini une fonction à appeler après une identification, on appelle cette fonction s'il y a eu succès, on signale l'erreur s'il y a eu échec. La fonction appelée pourra ensuite décider de passer à la page suivante si l'on est dans `xulforum.xul`.

POUR ALLER PLUS LOIN **Doc C**

Si vous voulez pousser l'expérience LDAP encore plus loin, la documentation du C SDK est à votre disposition, sur le site de Mozilla :

► <http://www.mozilla.org/directory/csdk-docs/>

ATTENTION **Bind anonyme**

L'identification LDAP ne se fera que dans la première page de XUL Forum, la page d'identification. Dans la seconde, c'est-à-dire `index.xul`, nous nous connecterons de manière anonyme au serveur LDAP (il n'y aura donc pas d'identification) pour seulement effectuer un rapatriement de données.

Autrement dit, pour `xulforum.xul`, seule la variable :

`gFonctions.apresIdentification()`

sera définie ; pour `index.xul`, seule :

`gFonctions.apresRecherche()`

sera utilisée.

Obtenir la liste des membres

Analyse du côté LDAP

La deuxième utilisation consiste à effectuer une recherche, sur tous les utilisateurs de XUL Forum par exemple. Le principe est le même que pour l'identification : d'abord, obtenir une opération. Ensuite, analyser les messages qui arrivent via LDAPListener.

L'appel à la fonction chercher dans le code précédent n'a pas été détaillé : nous allons maintenant le faire, car après l'identification, la voie est libre pour lancer une recherche !

```
function chercher() {
    var p = new Array();
    var c = new Object();
    gLdapURL.getAttributes(c, p);

    obtenirOp();
    gLdapOp.searchExt(gLdapURL.dn, gLdapURL.scope,
        gLdapURL.filter, c, p, 10, 10);
}
```

Cette partie est peu complexe : elle obtient juste une opération et lance une recherche suivant le DN, l'étendue (SCOPE_SUBTREE, c'est-à-dire le DN demandé et ses sous-éléments, nous l'avons choisi à l'initialisation), le filtre (objectClass=*), le nombre d'attributs cherchés, leurs valeurs (DN, CN...), une éventuelle limite de temps, puis de taille.

La partie la plus intéressante se concentre dans le listener, lorsque les résultats de la recherche arrivent.

Gestion des résultats de la recherche

```
case Components.interfaces.nsILDAPMessage.RES_SEARCH_ENTRY :
    dump("\n---ENTRY\n");

    try {
        var count = new Object();
        var attributs = pMsg.getAttributes(count); ❶
        var p = new Object();

        for (a in attributs) {
            if (attributs[a] == "jpegPhoto") { ❷
                var bcount = new Object();
                var binaires = pMsg.getBinaryValues("jpegPhoto",
                    bcount); ❸
                var l = new Object();
                var r = binaires[0].get(1); ❹
                dump("    jpegPhoto : "+l.value+" bytes\n");
            }
        }
    }
}
```

attributs est une liste : cn, dn, sn, etc. de toutes les propriétés disponibles

Indice 0 : il n'y a qu'une photo.

```

        var data = "";
        for (a in r) {
            data += String.fromCharCode(r[a]); ❸
        }
        p["jpegPhoto"] = btoa(data); ❹
    } else {
        var account = new Object();
        p[attributes[a]] = pMsg.getValues(attributes[a],
            account); ❺
    }
}
if (gLdapFonctions.apresRecherche != null)
    gLdapFonctions.apresRecherche(p); ❻

} catch (e) {
    dump(e+"\n");
}
dump("\n    "+pMsg.dn+"\n");
break;

case Components.interfaces.nsILDAPMessage.RES_SEARCH_RESULT :
    dump("\n---RESULT\n");
    dump("    Recherche terminee\n");
    break;

```

C'est ici que se concentre le cœur du traitement LDAP. Lorsqu'une entrée correspondant à la recherche arrive, la première chose à faire est de voir les différents attributs que porte cette entrée. ❶ Ils sont stockés dans la variable `attributes` et `count` porte le nombre d'attributs (pour une éventuelle boucle `for` avec un compteur au lieu de `for in`). La variable `p` sera un objet, rempli comme un tableau, avec dans les clés le nom de l'attribut et pour valeur la valeur de l'attribut correspondante. Nous la transmettrons à une éventuelle fonction indiquée par le script principal.

On parcourt donc les différents attributs. Si c'est une photo ❷ au format JPEG, on lui applique un traitement particulier. On récupère un composant LDAP enveloppant les données binaires ❸ implémentant `nsILDAPBERValue`, puis on demande à obtenir les données brutes de ce composant ❹. On parcourt les données et pour chaque octet, on le transforme en son caractère correspondant pour former une chaîne ❺. Cette chaîne est ensuite encodée en base64 ❻ et ajoutée au tableau de retour `p`. Si c'est un attribut normal (nom, e-mail, etc.) au format texte, on récupère sa valeur normalement ❼, tout en l'ajoutant au tableau de retour.

Enfin, si une fonction à appeler est définie ❽, on l'appelle en lui fournissant le tableau contenant les différents attributs de l'entrée obtenue.

◀ Hop : tout en base64 !

◀ `account` contient maintenant le nombre de valeurs pour l'attribut : par exemple 3 si `objectClass=top`, `objectClass=person`, `objectClass=inetOrgPerson`

B.A.-BA Base64

Elle permet de convertir des données binaires sous forme de chaîne qui ne risque pas d'être corrompue entre les différents encodages. Ainsi, si vous essayez d'ouvrir un exécutable avec un éditeur de texte, vous verrez une suite de caractères cabalistiques, qui dépendent de l'encodage dans lequel on considère le fichier.

La base64 s'affranchit de ces limites en encodant des données binaires en utilisant 64 caractères : de `a` à `z`, de `A` à `Z`, de `0` à `9`, `+`, `/` et `=` qui est un caractère spécial. La taille est d'environ 33% supérieure à des données non encodées.

La base64 est principalement utilisée pour l'encodage des pièces jointes dans les courriers électroniques. Elle est décrite par plusieurs RFC. La fonction JavaScript permettant l'encodage est `btoa` (« binary to ascii ») et celle permettant le décodage est `atob`.

POUR ALLER PLUS LOIN Certificats

Le traitement de type binaire n'est en général pas aussi complexe. Il est utilisé en interne par Mozilla pour le code de l'application mail : le composant enveloppant les données binaires d'un certificat est directement passé à un autre composant cherché de vérifier son authenticité.

ATTENTION Référence XULPlanet

Dans ce cas exceptionnel de composants XPCOM LDAP, la référence de XULPlanet indique par exemple :

```
void getBinaryValues (char* attr, out PRUint32 count,
    retval nsILDAPBERValue values)
```

En fait le prototype correct est :

```
nsILDAPBERValue values getBinaryValues (char* attr,
    out PRUint32 count)
```

Traitement XUL

Il faut maintenant exploiter les données analysées avec XUL.

Le traitement DOM pour remplir la liste des connectés

Utilisée avec une recherche sur toute l'organizational unit des utilisateurs : pour chaque entrée, la fonction ajoute un listitem à la listbox.

```
function remplirListeMembres(pAttributs) {
    if (pAttributs["cn"] == null)
        return;
    var l = document.getElementById("xf-index-membres-liste");
    var li = document.createElement("listitem");
    li.setAttribute("label", pAttributs["cn"]);
    li.setAttribute("class", "listitem-iconic");
    li.setAttribute("style", "list-style-image:
        ➡ url(\"chrome://xulforum/skin/stock_person.png\")");
    li.setAttribute("onclick",
        "remplirInfosMembre('"+pAttributs["uid"]+"')");
    l.appendChild(li);
}
```

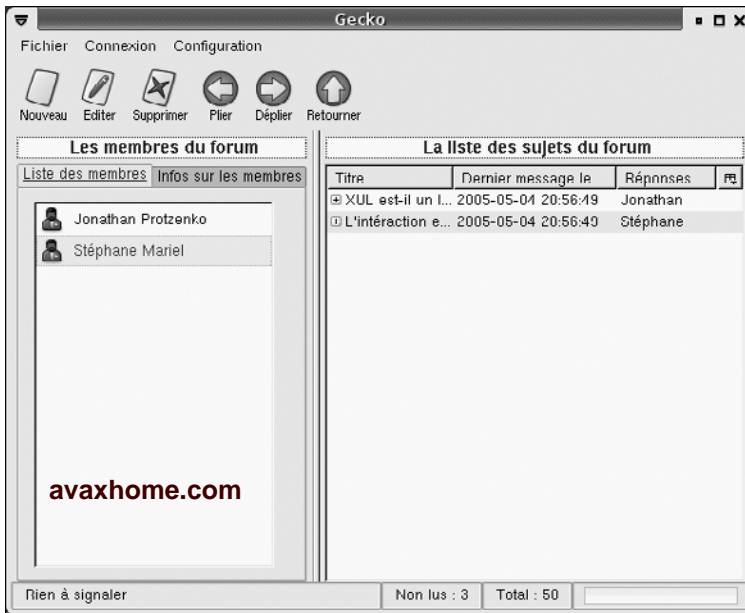
La fonction est très linéaire : on annule si c'est un nœud que l'on traite (comme ou=utilisateurs, o=xulforum). Si au contraire c'est un personne définie par son Common Name (CN), on continue le traitement.

On obtient l'élément <listbox> contenant la liste des usagers, on crée de toutes pièces un élément fils <listitem>, destiné à être ajouté à la <listbox>, puis on précise certains de ses attributs :

- le label, ou titre s'affichant sur l'item sera le nom de l'entrée LDAP (son common name) ;
- la classe sera listitem-iconic, faisant écho à menuitem-iconic, vu dans les premiers chapitres ;

- la propriété CSS `list-style-image`, déjà rencontrée maintes fois, sera celle d'une image représentant un petit bonhomme ;
- en cas de clic, il faudra afficher sur l'autre onglet du panel les informations du membre correspondant au `listitem` cliqué : on obtient au final une propriété `onclick` ressemblant à :
`onclick="remplirInfosMembre('jonathan')"`.

Le résultat est au final assez joli :



ALTERNATIVE Source en RDF

Il est bien sûr possible de remplir cette liste grâce au contenu d'une source RDF. Ceci facilite énormément le traitement. On peut même imaginer un script PHP se connectant au serveur LDAP et créant la source RDF, pour plus de simplicité. Un exemple de ce type existe sur le site du projet (voir `www/rdf.php` dans les sources). Pour la version web ne faisant pas appel à LDAP, la source sera bien sûr remplie grâce à RDF, mais du RDF créé à partir du contenu de la base MySQL.

Les informations d'un connecté

La fonction `remplirInfosMembre` ne fait que réutiliser les mécanismes vus précédemment : elle relance une recherche LDAP sur le membre en particulier dont on veut les informations, sur le serveur précisé dans la configuration et indique que lorsque les résultats de la recherche arriveront, ce sera la fonction `remplirPanneauInfosMembre` qui devra être appelée.

Cette dernière, à l'aide de DOM, remplit les différents attributs du panneau. Pour l'image, l'astuce est en fait d'utiliser ce qu'en anglais on appelle les *data url* : des images dont le contenu est directement spécifié dans l'attribut `src`. Il prend alors la forme `src="data:image/png;base64,contenuenbase64"` et ceci explique l'encodage en base64 précédent. Le type MIME est bien sûr au choix, nous utilisons dans notre cas le type `image/jpeg`.

Remplit les attributs d'un utilisateur en particulier c'est le retour sur la recherche sur le membre plus bas.

Callback lorsque l'on clique sur un utilisateur dans la liste : on lance une recherche LDAP sur les attributs de cet utilisateur en particulier.

Les informations seront remplies à l'appel du *callback* `remplirInfosMembre` vu plus haut : voici son contenu.

Remplir les attributs d'un utilisateur

```
function remplirPanneauInfosMembre(pAttributs) {  
  
    dump("Remplissage infos\n");  
    document.getElementById("xf-index-membres-avatar")  
        .src = "data:image/jpeg;base64,"+pAttributs["jpegPhoto"];  
  
    document.getElementById("xf-index-membres-localisation").value =  
        pAttributs["l"];  
    document.getElementById("xf-index-membres-email").value =  
        pAttributs.mail;  
    document.getElementById("xf-index-membres-description").value =  
        pAttributs.description;  
    document.getElementById("xf-index-membres-nom").value =  
        pAttributs.cn;  
}  
  
function remplirInfosMembre(pMembre) {  
    dump("Membre sélectionné : "+pMembre+"\n");  
    gLdapFonctions.apresRecherche = remplirPanneauInfosMembre;  
  
    var dn = "uid="+pMembre+","+gConfig.ldap.baseDn;  
    var hote = gConfig.ldap.hote;  
    initialiserLDAP(dn, hote);  
}
```

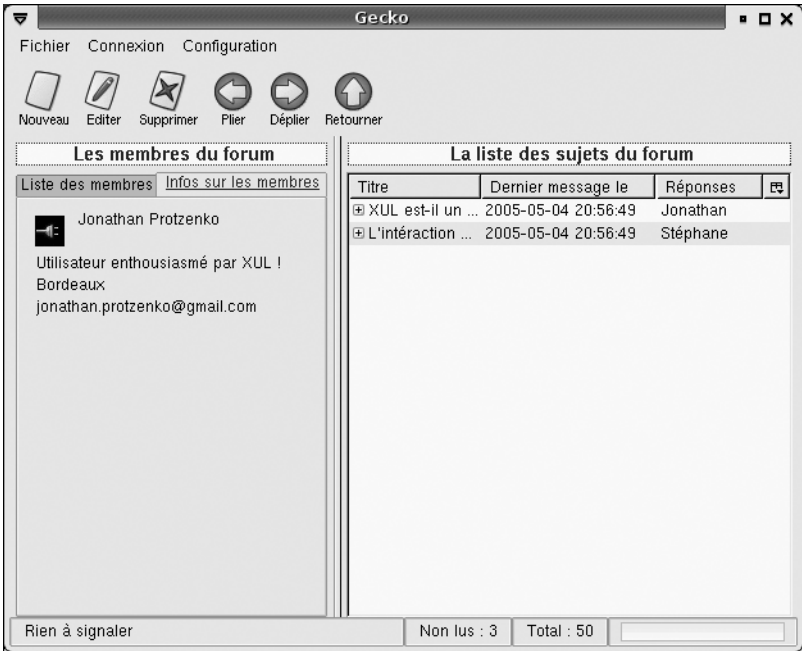


Figure 10-4
Le résultat final est plutôt réussi !

En résumé...

Ce chapitre peut paraître un peu obscur : en effet, si vous n'avez pas souvent eu l'occasion de travailler avec LDAP, les notions paraissent souvent trop abstraites pour être comprises clairement. Mais il n'est pas possible de détailler intégralement LDAP : les liens donnés en remarques et les « bonus » sur le site du projet vous permettront, si vous le désirez, de découvrir cette technologie souvent fort utile : vous pourrez même, si vous êtes conquis, gérer votre carnet d'adresses avec LDAP et le consulter en utilisant Thunderbird/Mozilla !

Pour ceux qui ont déjà utilisé LDAP, ce chapitre aura permis d'approfondir la création de composants XPCOM plus qu'obscurs (il aura fallu chercher certaines constantes dans des fichiers .h de LXR !). Peut-être la preuve aura-t-elle été faite que XUL est bel et bien un outil digne d'un environnement professionnel et pas seulement un gadget de « script kiddies » pour développeurs web en manque d'originalité.

POUR ALLER PLUS LOIN **Autres composants XPCOM**

Nous avons vraiment vu tout un pan des composants XPCOM de Mozilla dans ce chapitre. Il en existe de nombreux autres ! Si vous voulez vous essayer à d'autres extensions XPCOM, n'hésitez pas, plongez dans la référence de XULPlanet, essayez de trouver un exemple fonctionnel dans les sources de Mozilla et améliorez-le !

► <http://xulplanet.com/references/xpcomref/>

POUR ALLER ENCORE PLUS LOIN **Créez vos propres composants XPCOM**

Le titre parle de lui même : jetez donc un œil au livre dédié à ce sujet sur Mozilla.org (connaissances en C++ requises).

► <http://www.mozilla.org/projects/xpcom/book/cxc/html/index.html>

chapitre 11



Do-it-yourself widgets : XBL

Nous touchons maintenant au cœur de XUL : nous allons dans ce chapitre créer nos propres widgets, en mélangeant DOM, XML, XUL, HTML, JavaScript... en fait un résumé de ce que nous avons vu dans les chapitres précédents, agrémenté de quelques améliorations.

SOMMAIRE

- ▶ Introduction à la nouvelle technologie et présentation de notre widget
- ▶ Construction du binding fenetreMsg
- ▶ Intégration avec la fenêtre principale

MOTS-CLÉS

- ▶ binding XBL
- ▶ <content>
- ▶ <implementation>
- ▶ <handlers>
- ▶ champs et propriétés d'un widget
- ▶ modèle événementiel du DOM

La première question qui vient à l'esprit du programmeur est : « pourquoi refaire ce qui est déjà existant et créer nos propres widgets ? ». Cette interrogation, légitime, est en fait le fruit d'une mauvaise information. Lorsque nous créons nos propres widgets, à moins de devenir volontairement de mauvais programmeurs, nous ne réinventerons pas la roue. Un exemple trivial serait celui d'une boîte titrée. Au lieu d'écrire :

```
<label value="XUL Forum" /><vbox>...</vbox>
```

on pourra, à l'aide de XBL, écrire :

```
<box class="boiteTitree" titre="XUL Forum">...</box>
```

XBL est en fait de la réutilisation de code déjà existant, dans le but de centraliser en un « objet » XUL ce qui serait sans XBL éclaté sur plusieurs fichiers. Ainsi, notre widget prendra en charge les événements engendrés par son contenu, le changement/l'obtention de propriétés (sa position, son titre, etc.), il fournira des méthodes pour modifier son apparence, il définira son contenu, etc. Mais tout ceci n'est pas très parlant ; nous allons de suite plonger dans le cas concret : le widget `fenetreMsg`, abréviation de « fenêtre de message ».

CULTURE XBL dans Mozilla

En fait, de nombreux widgets sont des widgets XBL : le `progressmeter` en est un exemple. Sa déclaration CSS (que nous verrons quelques pages plus loin pour `fenetreMsg`) est la suivante :

```
progressmeter[mode="undetermined"] {
  -moz-binding:
    url("chrome://global/content/bindings/progressmeter.xml#
    ➤ progressmeter-undetermined");
}
```

Fonctionnement d'un widget XBL

Le widget `fenetreMsg`

Pour ne pas surcharger l'interface, nous allons utiliser un mélange de HTML, de XUL et de JavaScript pour créer des fenêtres volantes, sortes de cadres qui se déplacent « par-dessus » la page `index.xul`, que l'on peut réduire lorsqu'elles prennent trop de place et même masquer si le besoin s'en fait sentir. Une fois n'est pas coutume, nous allons d'abord montrer le résultat final avant même de commencer à programmer : voici une copie d'écran montrant deux `fenetreMsg`, avec des sujets d'exemple, l'une réduite et l'autre agrandie.

RAPPEL DOM Inspector

N'oubliez surtout pas de vérifier la hiérarchie DOM avec l'inspecteur. Il se révélera une aide très utile à la compréhension de la « logique XBL ».

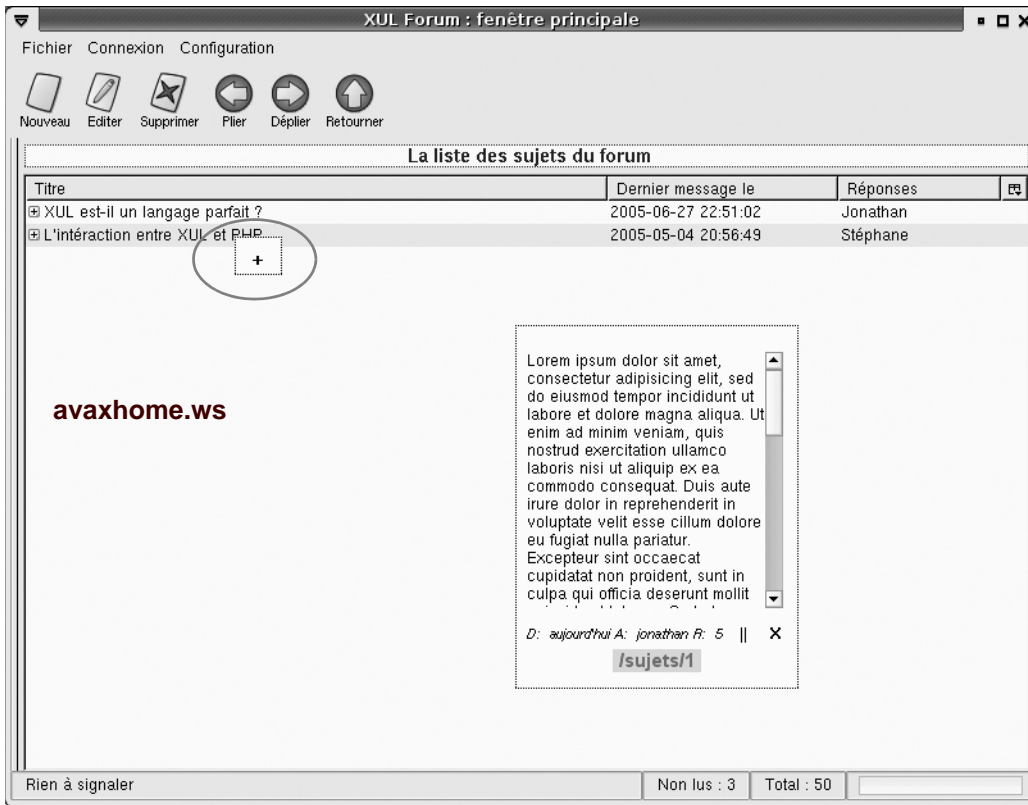


Figure 11-1 Un sujet déplié est visible et l'autre plié (le carré jaune avec le signe « + »)

Le panneau de gauche a été masqué pour plus de visibilité. Les sujets sont déplaçables par glisser/déposer en sélectionnant leur titre (un peu comme la barre supérieure d'une fenêtre dans un gestionnaire de fenêtres, ou *window manager*, classique), ou bien en maintenant les touches Alt et Shift enfoncées et en effectuant un glisser/déposer sur n'importe quelle partie de la fenêtre. Un double-clic sur la croix ferme la fenêtre, un double-clic sur le titre la réduit. Une fois réduite, elle prend l'allure d'un rectangle jaune avec seul le signe + au milieu ; un double-clic sur le + réouvre le sujet.

L'idée originale a été trouvée sur la suite de tests pour XBL des versions de Mozilla avec les fonctionnalités de débogage activées (*nightlies* notamment). Le concept de base a été largement amélioré : la réduction des fenêtres a été ajoutée, la fermeture fonctionne différemment (de même que l'ajout de fenêtres), le déplacement avec Alt + Shift n'était pas dans l'original et, amélioration notable, les fenêtres ne sont pas enfermées dans un tableau (*board*) qui limiterait leurs déplacements.

► <http://www.mozilla.org/projects/xbl/test5/test.html>

ASTUCE Tester le widget séparément

Durant ce chapitre, nous allons progressivement mettre en place les différents constituants du widget `fenetreMsg`. Pour ne pas s'enliser dans les fichiers principaux de l'application déjà complexes, il est plus judicieux de créer une page de « test », `testxbl.xul`, pour les widgets XBL. Nous les intégrerons ensuite à `index.xul`.

ALTERNATIVE Pop-up HTML/XUL

Cette technique est très impressionnante du point de vue des fonctionnalités, mais elle se limite à des versions récentes de Firefox/Mozilla (elle fonctionne sur les premières versions alpha et bêta de Mozilla 1.8, mais pas sur les versions stables telles que Firefox 1.0.x). Le mélange hasardeux HTML/XUL et l'utilisation d'attributs CSS `position: absolute` expliquent ce manque de compatibilité descendante. À l'heure de la parution de ce livre, il est probable que Firefox sera au moins en bêta, au mieux en version finale 1.5.

Que faire donc ? Si vous voulez tester le fonctionnement directement, téléchargez une version 1.5 de Firefox (les alpha sont très stables, mais réservées aux développeurs) et allez à la fin du livre pour voir comment développer avec la nouvelle organisation du dossier chrome (les explications sont en annexe). Si vous êtes bloqué à une version 1.0.x, l'écriture d'une fenêtre pop-up en HTML/XUL sera un très bon exercice supplémentaire ! Et peut-être même prétexte à un pop-up XBL...

Notre implémentation : le binding `fenetreMsg`

Dans cette partie, nous allons nous intéresser au contenu global du fichier `chrome://xul/forum/content/xbl/bindings.xml`. C'est ici que sera décrite l'intégralité du widget `fenetreMsg`. Le fichier porte l'extension XML mais aurait tout aussi bien pu s'appeler `bindings.xbl`. Nous verrons son contenu plus en détail dans les parties suivantes.

Nonobstant l'angoisse de la page blanche, nous allons dès maintenant commencer à remplir `bindings.xml`.

Un aperçu global du futur `bindings.xml`

```
<?xml version="1.0" ?>
<bindings id="xfBindings" xmlns="http://www.mozilla.org/xbl"
  xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:xbl="http://www.mozilla.org/xbl">
  <binding id="fenetreMsg">
    <content>
      <!-- ici le contenu du widget -->
    </content>
    <implementation>
      <!-- méthodes, propriétés, champs... -->
    </implementation>
    <handlers>
      <!-- gestionnaires d'événements -->
    </handlers>
  </binding>
</bindings>
```

Comme tout fichier XML qui se respecte, `bindings.xml` arbore le prologue XML traditionnel en première ligne. Plus bas, nous rencontrons l'élément `bindings`, qui sera amené à contenir un ou plusieurs éléments `binding`.

Dans notre cas, il n'y en a qu'un seul, c'est le binding portant l'ID `fenetreMsg`. Il contiendra trois éléments fils : `content`, `implementation` et `handlers`. Vous remarquerez qu'il y a de nombreux attributs `xmlns` sur le tag principal `bindings` : `xbl` (par défaut), `html`, `xul`, de nouveau `xbl`... en fait, on doit être capable de spécifier à chaque instant à quel espace de nommage appartient un élément, ou même, une propriété. Nous serons en effet amenés à utiliser des propriétés XBL sur des éléments XUL : `<xul:widget xbl:propriété="..." />`. Quant aux autres éléments comme `content`, `binding`, leur espace de nommage sera XBL par défaut.

Le contenu du widget : `<content>`

Le widget vu de l'extérieur : un bloc `div` et une classe CSS

Tout commence par le fichier `testxbl.xul` servant à tester le widget. Il contiendra les éléments de base d'une fenêtre, avec les espaces de nommage XUL et HTML, puis un bloc `div` particulier que nous allons voir immédiatement.

Le contenu de la fenêtre de test (que nous adapterons ensuite à `index.xul`)

```
<?xml version="1.0" ?>
<?xml-stylesheet href="chrome://xulforum/skin" type="text/css" ?>
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  ➔ there.is.only.xul"
  xmlns:html="http://www.w3.org/1999/xhtml"> ❶
  <html:div> ❷
    <html:div xmlns="http://www.w3.org/1999/xhtml"
      class="fenetreMsg" titre="Titre du sujet #1" id="sujet1"
      auteur="Stéphane" date="28/06/2005" reponses="5"> ❸
      <p>
        <b>Lorem</b> ipsum dolor sit amet, ... ❹
      </p>
    </html:div>
    <html:div xmlns="http://www.w3.org/1999/xhtml"
      class="fenetreMsg" id="sujet2" date="30/06/2005"
      auteur="Jonathan" reponses="1"> ❺
      <p>
        Lorem ipsum dolor sit amet, ...
      </p>
    </html:div>
  </html:div>
</window>
```

Il faut d'abord placer l'attribut `xmlns` correct pour pouvoir utiliser des éléments HTML au milieu d'une fenêtre XUL ❶. Ensuite, un bloc `div` général contiendra tous les widgets `fenetreMsg` ❷. Pourquoi placer tous

CULTURE Les bindings internes

Vous pourrez retrouver de nombreuses déclarations CSS transformant des widgets XUL en widgets XBL dans :

► <chrome://global/skin/global.css>

les widgets XBL dans un conteneur <div> supplémentaire ? Pour éviter une erreur de placement de Mozilla lors du glisser/déposer, erreur excusable au regard des manipulations extrêmes de DHTML/XUL que nous lui imposons. On crée ensuite un bloc div appartenant à la classe CSS `fenetreMsg` : on lui assigne ainsi une déclaration CSS spéciale, qui déclare son appartenance au binding `fenetreMsg`. Ce div sera notre widget `fenetreMsg` : il n'y a pas à proprement parler de widgets `fenetreMsg`, juste des éléments qui se transforment en widget XBL grâce à la déclaration CSS adéquate ③.

`FenetreMsg` possède également des attributs titre, auteur, réponses, date, etc. Ses éléments fils représentent le contenu HTML du sujet, placé dans un bloc p ④. Le bloc div possède l'attribut `xmlns="http://www.w3.org/1999/xhtml"`, ce qui indique que tous ses éléments fils seront des éléments HTML. Ceci évite de récrire le préfixe `<html:` à chaque nouvel élément fils.

Enfin on crée un deuxième bloc div ⑤ pour faire un deuxième sujet et par conséquent une deuxième fenêtre. Il faut maintenant modifier `xulforum.css`, de manière à prendre en compte la déclaration XBL. C'est l'attribut `-moz-binding` qui sera utilisé. Il prend la nature d'une URL, suivie d'un # indiquant l'ID du binding, un peu comme les ancres en HTML.

Les extensions CSS de Mozilla permettent d'assigner un binding à un élément normal

```
/* xbl */
.fenetreMsg {
  -moz-binding : url('chrome://xulforum/content/xbl/
    ➔ bindings.xbl#fenetreMsg');
}
```

En fait, nous aurions pu assigner la classe `fenetreMsg` à n'importe quel élément XUL ou HTML, puisque du moment qu'on lui applique un `-moz-binding`, il acquiert les propriétés du widget `fenetreMsg`. Il faut bien comprendre que l'on n'écrit pas `<fenetreMsg>` dans le code XUL. On utilisera un conteneur quelconque (généralement `<box>` ou `<div>`, plus adapté à du contenu HTML), qui sera là pour fournir un squelette, à compléter par le widget `fenetreMsg` dont nous allons voir la nature dans le paragraphe suivant.

Le contenu intérieur : mélange HTML et XUL

Le contenu final du widget sera déterminé grâce à un mélange savamment dosé, acceptant comme ingrédients :

- le contenu du widget `fenetreMsg` défini dans `bindings.xml` ;

- le contenu du widget support défini dans `index.xul` (ou `testxbl.xul` pour l'instant) ;
- les règles définissant la répartition des deux éléments, leur ordre, etc., placées dans `bindings.xml`.

Le contenu du widget XBL (bindings.xml)

```
<content>
  <html:div>
    <children />
    <html:div>
      <xul:description class="infosSujet" value="D: " />
      <xul:description class="infosSujet"
        xbl:inherits="value=date" />
      <xul:description class="infosSujet" value="A: " />
      <xul:description class="infosSujet"
        xbl:inherits="value=auteur" />
      <xul:description class="infosSujet" value="R: " />
      <xul:description class="infosSujet"
        xbl:inherits="value=reponses" />
      <xul:description value=" || " />
      <xul:description class="croixQuitter" value="X" />
      <html:center>
        <xul:description class="titreSujet"
          value="Sujet sans titre" xbl:inherits="value=titre" />
      </html:center>
    </html:div>
  </html:div>
</content>
```

- ◀ C'est le `div` principal dont on change les propriétés `top` et `left` ; il correspond à : `document.getAnonymousNodes(this)[0]`
Ceci est un rappel utile pour plus loin.

Le comportement n'est pas sans rappeler celui des overlays. D'abord, il y a le widget de base, portant l'attribut `class="fenetreMsg"`. Ses éléments fils (`<p>` notamment) sont supprimés et remplacés par le contenu de la balise `<content>`. Ensuite, un nouveau remplacement est effectué. La balise `<children>` est remplacée par la balise `<p>` (les enfants) du premier `<div>`. On a finalement :

```
<div>
  <div class="fenetreMsg">
    <div>
      <p><b>...</b>...</p>
      <div><xul:description />...</div>
    </div>
    <div class="fenetreMsg">
      ...
    </div>
  </div>
```

- ◀ Le conteneur de `testxbl.xul`.
- ◀ Premier widget `fenetreMsg`.
- ◀ Appartient à `<content>`.
- ◀ Remplacent `<children />`.
- ◀ Défini dans `<content>` ; englobe toutes les infos sur le sujet en cours.
- ◀ Autre widget `fenetreMsg`.

Ceci est donc le résultat théorique qui permet de comprendre l’affichage final du widget. En revanche, si l’on utilise DOM (et c’est le problème qui va survenir dans quelques pages), du point de vue logique, l’enfant du `<div class="fenetreMsg">` est toujours `<p>`. Nous allons y remédier, mais avant, il faut expliquer les attributs `titre`, `date` sur le `<div>` et les `xml:inherits` sur les `<description>`.

L’attribut `xml:inherits` permet, une fois appliqué sur un widget XUL (ou HTML), de spécifier de quels attributs du widget « support » ce dernier doit hériter. Ainsi, pour la seconde balise `description`, la valeur de l’attribut `value` doit être héritée d’après la valeur de l’attribut `date` du widget `support`. Il en va de même pour les quatrième et sixième descriptions.

Pour le titre, une valeur est déjà assignée. C’est en fait une valeur par défaut, qui sera utilisée s’il n’y a pas d’attribut `titre` sur le widget `support` et écrasée dans le cas contraire.

ATTENTION Mélanger HTML et XUL

Nous décidons ici d’utiliser les deux technologies, imbriquées l’une dans l’autre. Il ne faut surtout pas se méprendre. Ici, l’utilisation d’éléments `<div>` permet d’arriver à nos fins (des fenêtres en glisser-déposer), là où des éléments XUL auraient mal interprété les positions absolues en CSS. Mais surtout, le contenu interne de ces éléments HTML est lui-même du HTML : la mise en forme des sujets ne peut pas se faire grâce à XUL (XUL est un langage de description d’interface, pas de mise en forme de texte). Lorsque l’utilisateur voudra poster un message, le HTML sera permis et remplira son rôle : mettre en forme du texte grâce à des couleurs, un soulignement, des effets de police, des listes...

Il faut donc bien garder à l’esprit que XUL est le langage structurel de description d’interfaces et que HTML sera utilisé uniquement dans une optique de présentation, de mise en forme de texte.

La mise en forme CSS

Maintenant que la structure du widget est définie, une mise en forme s’impose. CSS va encore une fois nous venir en aide. La plupart des propriétés vous seront familières, car déjà rencontrées dans de nombreux chapitres auparavant. Nous ne détaillerons que celles qui n’ont pas été rencontrées auparavant et qui sont vraiment nouvelles.

```
.fenetreMsg > p {
  width: 200px;
  height: 200px;
  overflow: auto;
}
```

```

.fenetreMsg > div {
  position: absolute;
  border: 1px dotted black;
  padding: 8px;
  background-color: lightyellow;
}

.fenetreMsg description.titreSujet {
  font-family: Arial, Helvetica, Sans-Serif;
  font-weight: bold;
  color: grey;
  background-color: rgb(227,227,199);
  font-size: larger;
  padding-left: 5px;
  padding-right: 5px;
  cursor: move;
  max-width: 180px;
}

.fenetreMsg description.infosSujet {
  font-size: x-small;
  font-style: italic;
  padding-left: 0;
  padding-right: 0;
  margin-left: 0;
  margin-right: 0;
}

.fenetreMsg description.croixMaximiser {
  font-weight: bold;
  font-size: larger;
}

.fenetreMsg description.croixQuitter {
  font-weight: bold;
}

```

Pour le paragraphe amené à contenir le texte du sujet et les éventuelles réponses, on choisit une taille fixe et une directive `overflow: auto`. Cette dernière indique que si le texte prend plus de place que 200 pixels × 200 pixels, il faut ajouter des barres de défilement pour le <p> (le moteur Gecko s'en charge automatiquement).

Le div à la bordure en pointillés et à la couleur de fond jaune est le premier enfant de <content>. C'est ce bloc div qui sera redimensionné et déplacé. Sa position absolue indique qu'il est « détaché » d'un éventuel conteneur et qu'il se positionne de manière libre sur l'espace qui lui est réservé (la zone client représentée par la taille de la page).

Le titre du sujet subit quelques modifications de style et arbore notamment un curseur `move`. Celui-ci correspond à une croix avec des flèches le plus généralement et indique qu'un déplacement est possible sur la barre de titre. Ce curseur apparaîtra quand la souris survolera le titre. Une largeur maximale lui est assignée : c'est la propriété `max-width`. En la cou-

plant avec l'attribut `crop="right"`, c'est le texte qui déborde à droite qui sera coupé et remplacé par « ... ». Enfin, les descriptions apportant des informations sur le sujet sont rendues assez discrètes, tandis que les croix servant à maximiser et quitter le sujet sont assez marquées.

Un widget qui s'anime : <implementation>

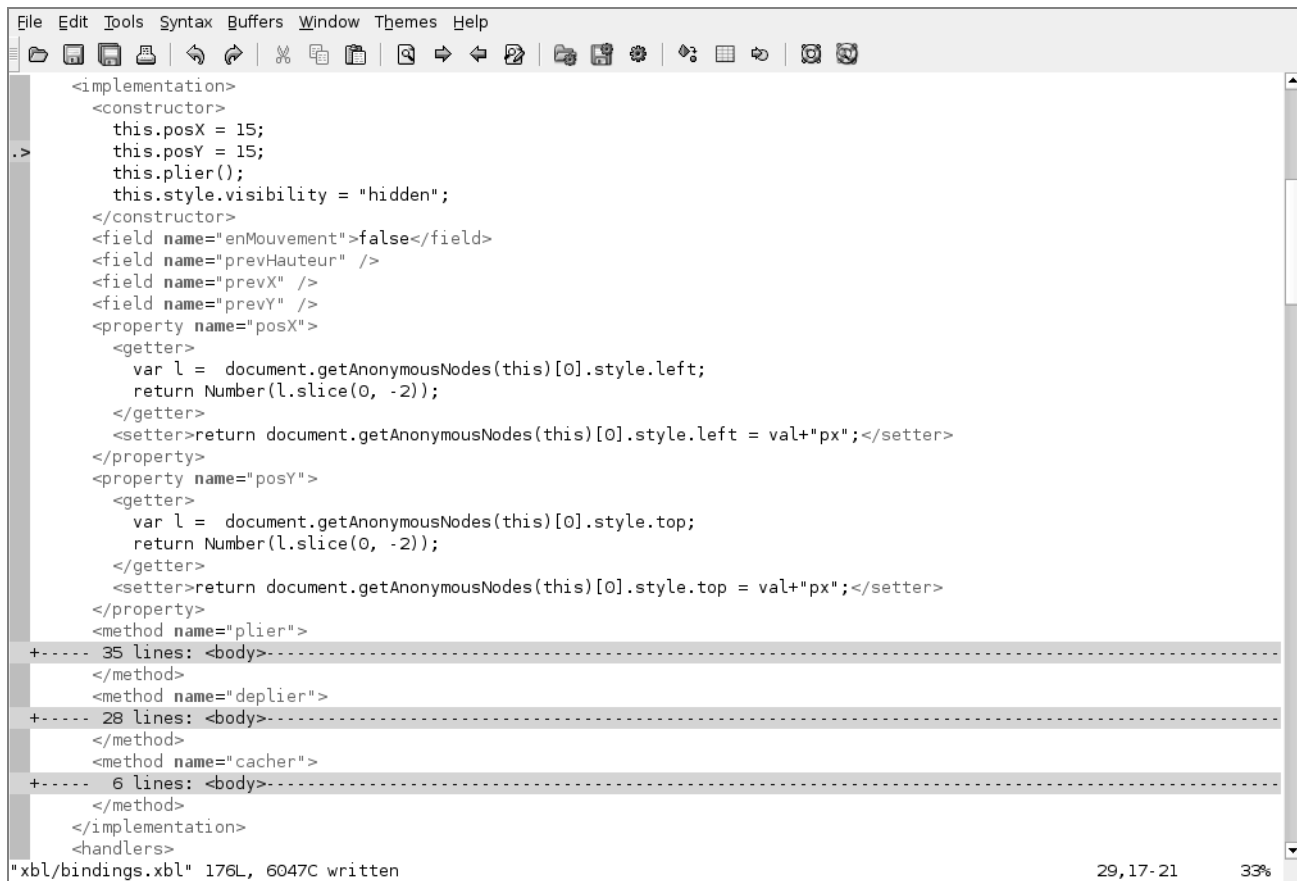
Les propriétés et les champs

Lors de la programmation du comportement de notre widget, nous serons amenés à définir des propriétés : l'objet est-il en mouvement ?, si on le déplie, quelle était sa hauteur précédente ?, quelle est sa position ? etc. Dans le dernier cas, elle se modifie grâce à des propriétés CSS (`left: "xxx px"; top: "xxx px";`). Plutôt que de manipuler ces propriétés à chaque fois (et ainsi ajouter le « px », le supprimer pour des calculs de déplacement qui impliquent des entiers...), nous allons définir deux propriétés `posX` et `posY`, qui se comporteront comme des nombres, mais en fait modifieront/rapatrieront les propriétés CSS ; la conversion de nombres en propriétés CSS avec « px » se fera de manière transparente.

Pour les propriétés `posX` et `posY`, on définit la manière de les obtenir grâce à <getter> et la manière de leur assigner une valeur grâce à <setter>. Pour l'obtenir, on récupère la valeur de la propriété CSS correspondante : c'est une chaîne. On en enlève les deux derniers caractères (on la coupe de la position 0 à la position -2, soit deux caractères avant la fin), on la transforme en nombre (car c'était une chaîne) et on la retourne. Pour lui assigner une valeur, on ajoute px à la variable `val` (la valeur fournie lorsque l'on écrit `monBinding.posX = 10` ; par exemple et qui est toujours appelée « val ») et on change la propriété CSS correspondante.

Cette abstraction des propriétés CSS pour ne manipuler que des entiers sera fort utile par la suite (dans les gestionnaires d'événements). Pour l'instant, vous pouvez en voir une utilisation dans le constructeur du widget (balise <constructor>) : pour tout widget nouvellement créé, on le positionne aux coordonnées (15;15), on le plie et on le masque. Lorsque nous en aurons besoin (nous verrons ceci lors de l'intégration à `index.xul`), nous le « démasquerons » puis nous le déplions.

Nous utilisons aussi des balises <field>, qui sont simplement des champs dans lesquels on peut stocker une variable (accessible par toutes les méthodes de l'objet) et qui ne nécessitent pas de traitement pour choisir/rapatrier leur valeur. Nous verrons leur utilité lors du détail des différentes méthodes.



```

<implementation>
  <constructor>
    this.posX = 15;
    this.posY = 15;
    this.plier();
    this.style.visibility = "hidden";
  </constructor>
  <field name="enMouvement">false</field>
  <field name="prevHauteur" />
  <field name="prevX" />
  <field name="prevY" />
  <property name="posX">
    <getter>
      var l = document.getAnonymousNodes(this)[0].style.left;
      return Number(l.slice(0, -2));
    </getter>
    <setter>return document.getAnonymousNodes(this)[0].style.left = val+"px";</setter>
  </property>
  <property name="posY">
    <getter>
      var l = document.getAnonymousNodes(this)[0].style.top;
      return Number(l.slice(0, -2));
    </getter>
    <setter>return document.getAnonymousNodes(this)[0].style.top = val+"px";</setter>
  </property>
  <method name="plier">
    <body>
    </method>
  <method name="deplier">
    <body>
    </method>
  <method name="cacher">
    <body>
    </method>
  </implementation>
</handlers>

```

"xbl/bindings.xbl" 176L, 6047C written 29,17-21 33%

Figure 11-2 Le contenu de la balise <implementation>

La variable `this` fait référence au widget `fenetreMsg`, le div portant l'attribut `class="fenetreMsg"`. On peut accéder à ses propriétés, ses champs, ses méthodes comme cela est fait dans le constructeur. Son élément fils est `<p>`, comme expliqué précédemment (et non pas `<div>`).

Les méthodes

Trois méthodes sont utilisées : elles se ressemblent beaucoup et permettent de plier, déplier et cacher la fenêtre de message.

ASTUCE Méthode prenant des paramètres

Ici nous ne définissons que le corps de la méthode (`body`). Si vous avez besoin de lui fournir des paramètres, vous pouvez placer le tag :

```
<parameter name="nomDuParamètre" />
```

avant le tag `<body>`, puis utiliser ensuite dans le code JavaScript la variable `nomDuParamètre`.

n est le contenu de <content>, n[0] est <content><div>.

Les enfants du <content><div>.

Évite les noeuds #text et les noeuds #comment qui n'ont pas de style.

Les noeuds contenus dans la balise <div> du fichier XUL principal, c'est-à-dire les <children />.

On crée la petite croix qui servira à maximiser plus tard dynamiquement et ce à chaque fois.

avaxhome

B.A.-BA display et visibility

Lorsque l'on veut masquer un élément, il y a deux possibilités. La plus rapide et la moins ennuyeuse est de mettre son attribut `visibility` à `hidden` (annulable en choisissant `visible`). L'élément n'est plus affiché sur la page, mais par contre, l'espace nécessaire est toujours alloué, d'où de grands espaces vides correspondant à l'élément non affiché. L'autre méthode consiste à mettre son attribut `display` à `none` (pour annuler, il suffit de lui attribuer une valeur vide). Ainsi, l'élément se comporte comme s'il avait été supprimé de la page. Mais lorsqu'on le « remet » dans la page, l'agencement peut être modifié. Testez donc toujours les deux méthodes pour savoir laquelle convient le plus à votre cas !

Plier

Comment plier une fenêtre ?

En masquant un par un ses éléments et en réduisant sa taille

```
<method name="plier">
  <body>
    <![CDATA[
      var n = document.getAnonymousNodes(this);
      this.prevHauteur = n[0].style.height;
      n[0].style.height="12px";

      var n1 = n[0].childNodes;
      for (var i = 0; i < n1.length; ++i) {
        if (n1[i].nodeName[0] != "#") {
          n1[i].style.display = "none";
        }
      }

      var n2 = this.childNodes;
      for (var i = 0; i < n2.length; ++i) {
        if (n2[i].nodeName[0] != "#") {
          n2[i].style.display = "none";
        }
      }

      var croix = document.createElement("description");
      croix.setAttribute("class", "croixMaximiser");
      croix.setAttribute("value", "+");
      n[0].appendChild(croix);
    ]]>
  </body>
</method>
```

Pour réduire la fenêtre, nous adoptons une procédure systématique : d'abord masquer un à un les enfants du premier <div> contenu dans <content> (lui n'est pas masqué, seuls ses enfants le sont) puis masquer les enfants du <div> de support (la balise <p>).

Pour obtenir les balises de <content>, on utilise la méthode `document.getAnonymousNodes` appliqué à l'objet `this`. Elle renvoie une liste : son premier élément est le premier enfant de <content>, c'est-à-dire le <div> qui nous intéresse. Il n'a qu'un seul enfant : un autre <div>, mais la boucle `for` laisse la voie libre pour ajouter d'autres éléments.

Ceci permet de résoudre le problème posé en début de chapitre, qui précisait que selon le DOM les enfants de `this` n'étaient que la balise <p>. Il est intéressant de noter l'emploi de la propriété `nodeName` d'un nœud DOM : contrairement à `tagName`, elle est définie pour tous les nœuds, même les nœuds texte et commentaire, ce qui permet d'ailleurs de les éviter en testant la nature du premier caractère du `nodeName`. Pour obtenir les enfants du <div> de support, on utilise l'objet `this` et sa propriété `childNodes`. On parcourt de la même manière tous ses enfants et on les masque un par un.

Quels sont les widgets restants ? Le `<div class="fenetreMsg">` de support mais ce dernier est purement structurel et le premier `<div>`, enfant de `<content>` : c'est celui qui a une bordure noire pointillée et un fond jaune (celui qui en réalité se révèle à l'affichage). Il est toujours visible : réduit à 12 pixels de hauteur, on lui ajoute une croix créée de toutes pièces ; elle servira à le maximiser plus tard. Le sujet est plié !

Déplier

Pour déplier, on rétablit la hauteur originale du cadre (ceci peut servir si l'on décide d'implémenter le redimensionnement plus tard) et on lance trois boucles.

- La première passe en revue les éléments fils du `<content><div>` et supprime purement et simplement la croix ayant servi à maximiser.

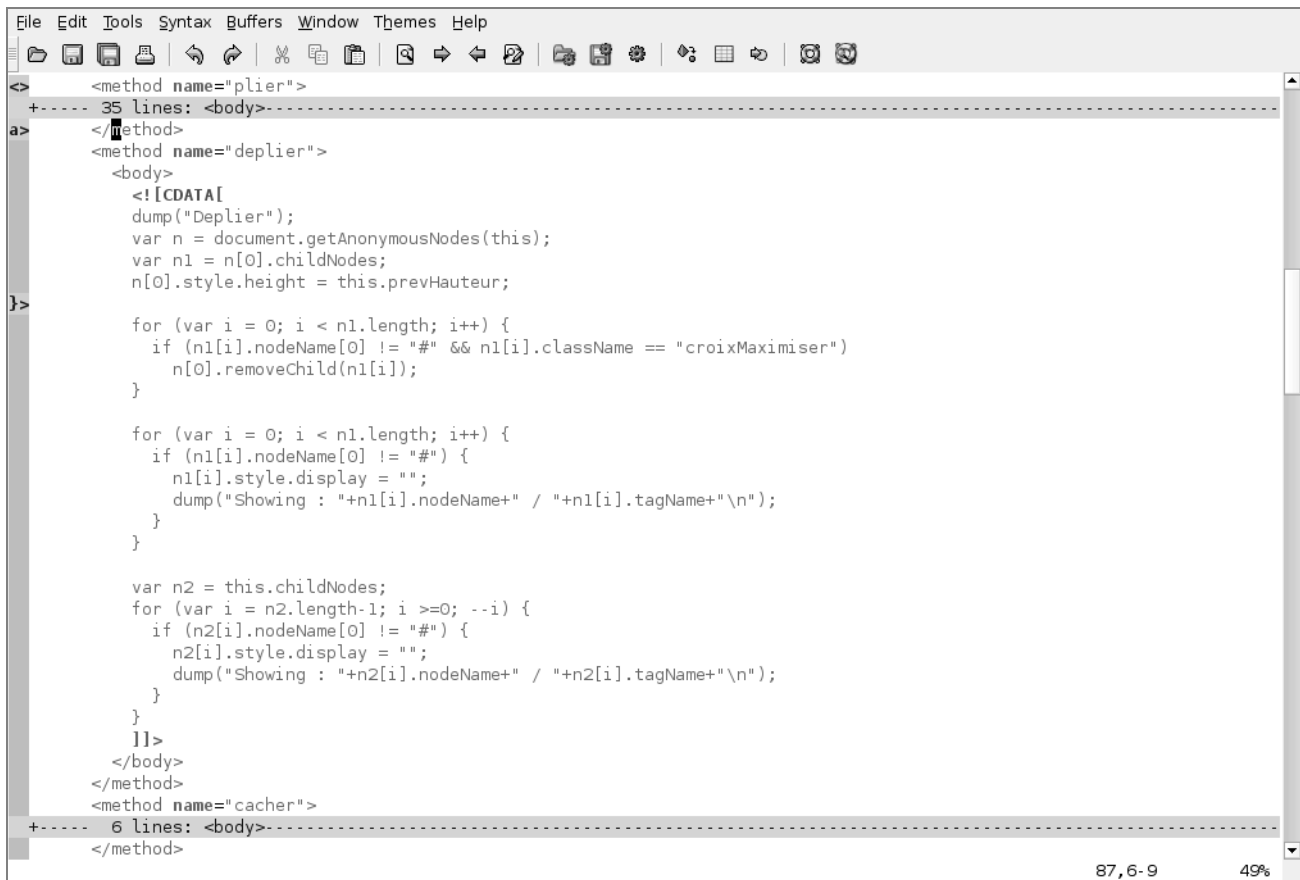


Figure 11-3 Le dépliage est encore plus simple !

- La seconde parcourt de nouveau les mêmes éléments, mais pour les montrer cette fois-ci.
- La dernière parcourt les éléments fils du `<div class="fenetreMsg">` de support et les montre eux aussi.

Cacher

La méthode servant à cacher un sujet (double-clic sur la croix) est simplissime :

```
<method name="cacher">
  <body>
    <![CDATA[
      this.style.visibility = "hidden";
      this.plier();
    ]]>
  </body>
</method>
```

Il suffit de masquer le sujet et de le plier. Cette astuce évite des erreurs lorsque l'on remontre l'objet (erreurs qui seraient arrivées si on avait utilisé la propriété CSS `display`) et le pliage évite l'apparition d'un curseur de déplacement fantôme, qui apparaîtrait si on passait la souris au-dessus du cadre de titre, non affiché, mais toujours présent.

Cependant, comment le widget XBL sait-il à quel moment déclencher le pliage, le dépliage ? Comment réagit-il aux événements qui se produisent en lui ? Nous avons vu les deux premiers tags présents dans un binding : `<content>` et `<implementation>`. Il nous reste le dernier à étudier : `<handlers>`.

Un widget réactif : le `<handler>`

C'est ici un autre avantage des widgets XBL qui se révèle : en plus de centraliser le contenu, les méthodes, ils centralisent aussi les événements. Toutes les méthodes `onXXX` que l'on aurait retrouvées dans un fichier JavaScript pour un widget classique se retrouvent sous une forme équivalente dans la balise `<handlers>` d'un widget XBL, qui contiendra ce que l'on appelle les gestionnaires d'événements. Un exemple simple illustrera de suite ce concept.

Le double-clic

Dans la balise `<handlers>` on retrouve une balise `<handler>` par événement

```
<handler event="dblclick">
  switch (event.originalTarget.className) {
    case "titreSujet" :
      this.plier();
      break;
    case "croixMaximiser" :
      this.deplier();
      break;
    case "croixQuitter" :
      this.cacher();
      break;
  }
</handler>
```

Ici, l'événement pris en compte est le double-clic. Les noms des événements sont les mêmes que ceux des attributs que l'on peut porter sur les éléments XUL, avec le on en moins.

B.A.-BA Les différents événements, liste complète

- `onblur/onfocus` : l'élément perd la focalisation ou gagne la focalisation ; utile pour des champs texte ;
- `onchange` : lorsque la valeur change, pour un champ texte, une liste déroulante, etc. ;
- `onclick, ondblclick` : clic et double-clic ;
- `onkeydown/onkeyup` : une touche est enfoncée ou relâchée ;
- `onkeypress` : après une pression sur une touche ;
- `onload/onunload` : le document (l'image) est chargé(e) ou déchargé(e) ;
- `onmousedown/onmouseup` : le bouton de la souris est baissé ou relevé ;
- `onmouseover` : la souris est sur l'élément ;
- `onmousemove` : la souris est déplacée ;
- `onmouseout` : la souris quitte l'élément ;
- `onreset/onsubmit` : le formulaire est remis à zéro ou envoyé (balise HTML `<form>`) ;
- `onresize` : la fenêtre est redimensionnée ;
- `onselect` : du texte est sélectionné ;
- `onerror/onabort` : le chargement d'une image a reçu une erreur ou est interrompu.

Le plus généralement, seuls quelques éléments à la souris seront utilisés, éventuellement les `on(un)load` sur un `<window>`.

Dans la majorité des cas, sur la référence de XUL Planet, outre les traditionnels `oncommand`, `onload`, etc., vous verrez quels sont les signaux supplémentaires disponibles pour un élément (comme l'élément `<textbox>` qui propose `onchange` et `oninput`).

► <http://xulplanet.com/references/objref/Event.html>

Chacun des trois éléments d'une fenêtre sur lesquels on peut double-cliquer (la croix pour quitter, le titre pour réduire, le + pour maximiser) est différencié des deux autres par sa classe CSS. On peut donc voir, lorsqu'un double-clic est effectué sur le `<div class="fenetreMsg">` ou l'un de ses enfants, à quelle classe appartient l'objet à l'origine du double-clic et adapter la réaction à adopter.

Lorsque l'on programme le gestionnaire d'événement, la variable `event` est créée automatiquement et représente l'objet `Event` associé à l'événement qui vient de se produire.

En effet, le modèle événementiel du DOM indique qu'une fonction de callback reçoit comme premier paramètre un objet `Event`. Ainsi en JavaScript on écrira, pour une fonction de callback traditionnelle :

```
function maFonctionDeCallback(e) {
    dump("J'ai un nouvel événement !\n");
    dump("Il s'est produit à l'origine sur une cible de type"
        +e.originalTarget.tagName+"\n");
}
```

L'objet `Event` possède plusieurs propriétés intéressantes. La première utilisée est `originalTarget`. Elle se réfère au nœud original qui a déclenché l'événement (le `xul:description` du titre par exemple) et diffère en cela de la propriété `target`, qui, elle, représente le nœud sur lequel l'événement a été dispatché (notre widget `fenetreMsg`).

Ainsi, on examine la classe du nœud sur lequel s'est produit à l'origine l'événement et l'on réagit en conséquence : le code parle de lui-même.

Le bouton de la souris est baissé

Nous allons successivement voir les handlers pour les différents événements qui se produisent lors du déplacement de la fenêtre et ce dans l'ordre chronologique.

Le premier événement qui se produit est `onmousedown` : le bouton de la souris est baissé.

```
<handler event="mousedown">
    var t = event.originalTarget;
    if ((t.tagName=="xul:description" && t.className ==
        "titreSujet") || (event.altKey && event.shiftKey)) { ❶
        this.enMouvement = true;
        this.prevX = event.clientX; ❷
        this.prevY = event.clientY; ❷
    }
    var max = 0;
    var divs = document.getElementsByTagName("div");
```

```

    for (var i = 0; i < divs.length; i++) {
      if (divs[i].style.zIndex > max)
        max = divs[i].style.zIndex; ❸
    }

    event.target.style.zIndex = Number(max)+1;
    document.getAnonymousNodes(this)[0].style.zIndex =
      Number(max)+1;
  </handler>

```

Nous ferons ici usage des champs vus plus haut mais dont l'utilité n'avait pas été expliquée.

Tout commence avec un test conditionnel ❶ : si l'événement s'est produit sur le titre, ou si les touches « alt » et « shift » ont été maintenues enfoncées (les mêmes propriétés existent pour les autres modificateurs : `ctrlKey`, `metaKey`), c'est un déplacement de la fenêtre qui a été lancé. L'objet `fenetreMsg` est donc en mouvement et sa position actuelle est stockée dans `prevX` et `prevY` ❷.

`clientX` et `clientY` sont deux propriétés d'un événement qui indiquent à quelle position a eu lieu l'événement (à la souris bien sûr), par rapport au coin supérieur gauche de la zone client (la zone de l'écran occupée par la page). Le défilement n'est pas pris en compte : pour cela il faudra utiliser `pageX` et `pageY`. Si vous voulez les coordonnées par rapport à l'écran, `screenX` et `screenY` vous seront utiles (pour ouvrir une fenêtre pop-up à l'endroit d'un clic de souris par exemple).

Ceci servira lorsque nous bougerons la souris : en calculant la différence de la nouvelle position (l'événement `onmousemove` advient après que la souris a été bougée) et de la position précédente (`prevX` et `prevY`), on obtiendra les coordonnées du vecteur déplacement du cadre sujet, qui seront appliquées à ses attributs `posX` et `posY` pour le déplacer. On pourra ensuite remplir `prevX` et `prevY` avec les coordonnées actuelles pour que le prochain `onmousemove` puisse calculer le nouveau vecteur déplacement.

Enfin, nous utilisons une propriété CSS un peu étrange : `zIndex` ❸. C'est la position du cadre `div` par rapport à l'axe vertical, c'est-à-dire de l'écran vers l'œil du lecteur. Une position supérieure aux autres indique que le cadre doit être placé « par-dessus » les autres. Pour que le cadre que nous déplaçons soit toujours au-dessus des autres, on examine tous les cadres du document et on récupère la valeur du `zIndex` maximal. On y ajoute 1, pour que notre cadre en déplacement soit au-dessus de tout autre cadre et on l'applique aux deux `div`, le `div` support et le `div` du `<content>`.

La souris est déplacée

```
<handler event="mousemove">
  if (this.enMouvement) {
    this.posX += event.clientX-this.prevX;
    this.prevX = event.clientX;
    this.posY += event.clientY-this.prevY;
    this.prevY = event.clientY;
  }
</handler>
```

Comme expliqué, on calcule la différence de la nouvelle position avec la précédente et on applique cette différence, correspondant au déplacement qui s'est produit, aux coordonnées du cadre, qui seront transformées automatiquement en attributs CSS grâce au `<setter>`. On prend ensuite les coordonnées actuelles comme coordonnées précédentes ce qui permettra de calculer de nouveau le déplacement au prochain événement `mousemove`.

Généralement, les événements `mousemove` se suivent à des intervalles très rapprochés (au grand maximum après un déplacement de trois pixels).

La souris est relâchée

Le handler prend une ligne et arrive juste après que l'on a relâché le bouton de la souris.

```
<handler event="mouseup">
  this.enMouvement = false;
</handler>
```

Maintenant que le bouton est relâché, le pop-up ne peut plus suivre les déplacements de la souris, autrement dit, le cadre sujet n'est plus en mouvement. C'est ce qu'indique cette ligne.

Mise en relation avec la page principale de XUL Forum

La totalité du contenu du fichier `bindings.xml` a été explicitée. Il reste maintenant une dernière formalité à accomplir : intégrer ce nouveau widget avec `index.xul` et le lier à l'arbre : un double-clic sur un sujet ou un message devra ouvrir un cadre correspondant et devra identifier le sujet ou le message à l'origine de ce clic, ce qui n'est pas forcément évident avec un arbre dont le contenu est créé par RDF (car il pose un problème similaire à celui des `<children />` au niveau du DOM).

Modifications dans le fichier XUL

Tout à la fin de `index.xul`, nous préparerons quelques sujets vides (qui se cacheront d'eux-mêmes grâce au constructeur XBL), par exemple cinq, ce qui permettra de les utiliser au fur et à mesure que le besoin s'en fait sentir et de ne pas dépasser un nombre « limite » de sujets, qui rendrait la navigation impossible.

```
<html:div id="xf-div-popups">
  <html:div xmlns="http://www.w3.org/1999/xhtml"
    class="fenetreMsg">
    <p></p>
  </html:div>
  <html:div xmlns="http://www.w3.org/1999/xhtml"
    class="fenetreMsg">
    <p></p>
  </html:div>
  ...
</html:div>
```

Ces sujets sont pré-remplis avec leur élément `<p>` amené à contenir le HTML du sujet : nous allons voir de suite comment tirer parti de cette pré-organisation avec JavaScript.

Modifications dans le JavaScript

La fonction qui gérera le double-clic sur l'arbre s'appellera `ouvrirPopupSujet`. Nous assignons un gestionnaire d'événement au double-clic sur l'arbre grâce à cette ligne de code, placée en tête de la fonction d'initialisation.

```
document.getElementById("xf-index-arbre").addEventListener(
  "dblclick", ouvrirPopupSujet, true);
```

ALTERNATIVE **Attribut** `ondblclick`

Nous aurions bien sûr pu utiliser :

```
<tree ondblclick="ouvrirPopupSujet(e)" />
```

Ceci n'est qu'un moyen alternatif qui peut néanmoins se révéler utile lorsqu'il faut attacher/détacher dynamiquement un gestionnaire d'événement.

La fonction `ouvrirPopupSujet` est la suivante :

On cherche si on a déjà utilisé un div pour ce sujet.

Si on n'en a pas trouvé, on prend le premier qui est libre (il est aussi libre s'il est caché)

La fonction ouvrirPopupSujet : la réponse à un double-clic sur l'arbre

```
function ouvrirPopupSujet(e) {
    var t = document.getElementById("xf-index-arbre"); ❶
    if (t.view.selection.count != 1)
        return; ❷
    var index = t.currentIndex;
    var col = t.columns["xf-index-arbre-id"];
    var titre = e.currentTarget.view.getCellText(index,
        col)+"\n"; ❸

    var divs = document.getElementById(
        "xf-div-popups").childNodes;
    var trouve = false;
    var i;

    for (i = 0; i < divs.length; ++i) { ❹
        if (divs[i].nodeName[0] != "#" && divs[i].id == titre) {
            trouve = true;
            dump(divs[i].id+"\n");
            divs[i].deplier();
            dump(divs[i].nodeName);
            break;
        }
    }

    if (!trouve) { ❺
        for (i = 0; i < divs.length; ++i) {
            if (divs[i].nodeName[0] != "#" && (divs[i].id == "" ||
                divs[i].style.visibility == "hidden")) {
                trouve = true;
                break;
            }
        }
    }

    if (!trouve) { ❻
        ajouterErreur(gStringBundle.getString("tropSujetsOuverts"));
        return;
    }

    var div = divs[i]; ❼
    div.id = titre;
    div.setAttribute("auteur", "jonathan");
    div.setAttribute("date", "aujourd'hui");
    div.setAttribute("reponses", "5");
    div.setAttribute("titre", titre.slice(22, -1));

    var p = div.getElementsByTagName("p")[0];
    p.innerHTML = "<b>Lorem ipsum</b> dolor sit amet,... ?<br /> "
        + titre; ❽
    div.appendChild(p);

    div.style.visibility = "visible";
    div.deplier(); ❾
}
```

La réponse à adopter lors d'un double-clic sur l'arbre commence par une ligne toute simple : il faut récupérer l'arbre que l'on va traiter ❶. S'il n'a aucune ligne sélectionnée, on arrête de suite le traitement ❷, car il n'y aurait aucun sujet à montrer.

Ensuite, les instructions sont moins évidentes. Commençons par la sixième ligne ❸. La propriété `currentTarget` de l'objet `Event` représente le nœud actuellement touché par l'événement (pas le nœud original : propriété `originalTarget` et pas le nœud sur lequel l'événement est survenu : propriété `target`). Ce nœud est bien sûr un élément `tree`. Si vous étudiez ses propriétés sur la référence XUL Planet, vous remarquerez tout en bas de la page une ligne `view ; Type: nsITreeView`. Cette propriété d'un `<tree>` est en fait un objet servant à créer le contenu de l'arbre. Il en existe un pour les arbres dont le contenu est directement spécifié dans le fichier XUL, un pour les arbres RDF, vous pouvez même créer vos propres `view` !

Nous ne détaillerons pas ce processus complexe (une URL pour aller plus loin vous est donnée en marge), mais nous retiendrons que parmi les nombreuses fonctions de l'interface `nsITreeView`, il en existe une appelée `getCellText`. Elle prend comme paramètres l'index de la ligne sélectionnée (obtenu grâce à la propriété `currentIndex` de l'arbre) et un élément XUL représentant la colonne dont on veut obtenir le texte. Nous l'utilisons pour obtenir le contenu de la colonne ID (voir astuce ci-dessous).

► <http://xulplanet.com/tutorials/xultreeview.html>

ASTUCE Colonne cachée

Nous avons ajouté une colonne cachée dans le document XUL `index-forum-overlay.xul`

```
<treecol id="xf-index-arbre-id" label="id" flex="1"
  hidden="true" ignoreincolumnpicker="true" />
</treecols>
```

La colonne est masquée et n'apparaît pas dans le « column picker » qui permet de déterminer quelles colonnes afficher. On ajoute ensuite dans `<treerow>` l'élément correspondant.

```
<treecell label="?message" />
```

Ceci permet de stocker pour chaque ligne son identifiant sous la forme d'un URI « `http://www.xulforum.org/sujets/XXidsujetXX/messages/XXidmessageXX` » unique, qui servira ensuite lorsque nous voudrions rapatrier le contenu d'un message particulier.

ajaxhome

La suite est plus conventionnelle : on parcourt la liste des blocs `div` déjà présents dans le document `index.xul` ❹. Si l'on trouve un bloc dont l'ID est le même que celui du message double-cliqué, on le déplie et on le montre. Si ce premier passage n'a rien donné, on refait un deuxième pas-

ATTENTION Compatibilité entre les différentes versions de Mozilla

Pour Gecko 1.8 (Mozilla 1.8 devenu entre temps Seamonkey 1.0, Firefox et Thunderbird 1.5), le deuxième argument est effectivement un élément `<treecol>`. En revanche, pour les versions inférieures (Mozilla 1.7.x, Firefox et Thunderbird 1.0.x), le deuxième argument est une chaîne donnant juste l'ID de la colonne. Attention donc !

sage ⑤ qui cherche un bloc `div` libre : un sujet qui serait soit masqué (l'utilisateur a quitté avec la croix), soit non encore utilisé. Si l'on n'a toujours pas trouvé de widget XBL de libre, on affiche un message d'erreur demandant de fermer un sujet ⑥.

La suite n'est valable que si l'on a trouvé un sujet libre ⑦ (instruction `return` dans le cas contraire). L'ID du widget est désormais l'ID du message à afficher, son titre, sa date, son auteur et son nombre de réponses sont aussi changés. Son élément fils `<p>` servira à contenir du HTML engendré pour afficher les différents messages : on utilise donc la propriété `innerHTML` qui permet de gagner du temps ⑧ en rentrant directement le contenu sous forme de HTML (et non pas sous forme d'éléments DOM, alternance de nœuds `#text` et de nœuds HTML).

Enfin, lorsque le sujet est prêt, on le déplie et on le montre ! ⑨

Nous pourrions au passage implémenter le comportement des trois derniers boutons de la barre d'outils : le principe est le même et nous ne ferons pas au lecteur l'affront de détailler les callbacks. Il suffit de parcourir les `div` comme nous l'avons fait plus haut et, pour chacun, d'appeler soit sa méthode `plier`, soit sa méthode `déplier`, soit sa méthode `masquer`.

En résumé...

Ce chapitre rompt la continuité des précédents : il n'a en effet pas utilisé de composants XPCOM ! Le plus dur est passé maintenant et l'on s'achemine vers les ultimes chapitres. Les dernières avancées en matière d'interface ont été définitivement validées :

- un nouveau langage, XBL, nous a permis de définir un contenu générique, implémentant ainsi un « nouveau widget » (par l'intermédiaire d'une classe CSS) ;
- ce widget possède ses propres propriétés, qui se modifient dynamiquement, ses méthodes, ses champs ;
- il est capable de réagir aux événements ;
- il est parfaitement intégré dans la page principale de l'interface.

Cependant et c'est la dernière limitation avant de rendre XUL Forum totalement fonctionnel, la liaison avec le serveur n'est pas établie en ce qui concerne les sujets. Nous n'avons pour l'instant aucun moyen d'envoyer un nouveau sujet sur le serveur, ni d'en rapatrier les différentes réponses.

Nous pourrions bien sûr utiliser une n-ième fois RDF pour créer le contenu des cadres de sujet, mais ceci ne résout pas le problème des communications dans le sens client vers serveur ! Nous allons donc voir une nouvelle méthode de communication dans le chapitre suivant : les services web, très en vogue à l'heure actuelle...

POUR ALLER PLUS LOIN Améliorer encore l'exemple du mini window manager

Ce type de programmation en HTML, XUL, JavaScript et CSS est assez amusant : on obtient vite un résultat très dynamique, très « animé », pour une interface qui semble parfois austère.

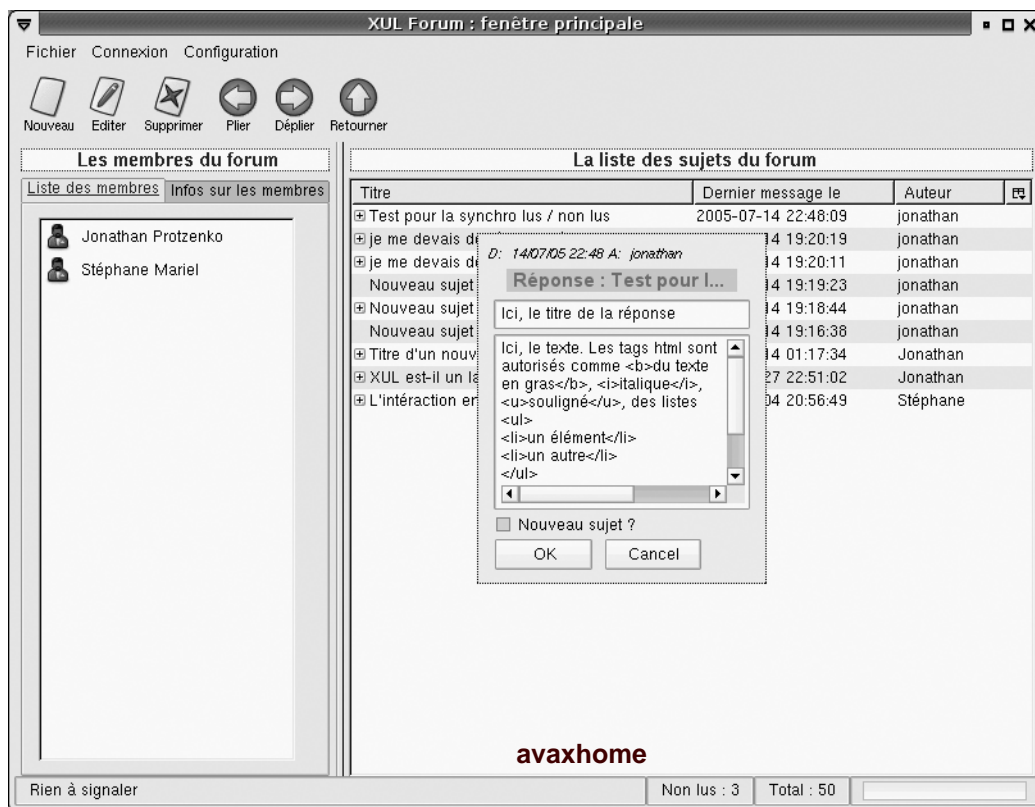
Une idée serait donc de pousser le concept de gestionnaire de fenêtres un peu plus loin : il serait ainsi possible d'ajouter une barre des tâches dans laquelle se réduiraient les sujets minimisés, d'implémenter un redimensionnement pour les cadres de sujet... bref de rendre ce concept totalement professionnel !

Ceci existe déjà et s'appelle Robin : le résultat est impressionnant techniquement. C'est un vrai gestionnaire de fenêtres en XUL, HTML, JavaScript et CSS !

► <http://robin.sf.net>

12

chapitre



Les services web, pour une communication harmonieuse entre client et serveur

En programmation, la communication est devenue essentielle. Nous travaillons dans un environnement connecté, où des humains ont besoin de communiquer avec des programmes, mais aussi un environnement où les programmes ont parfois besoin de communiquer entre eux !

SOMMAIRE

- Le serveur SOAP en PHP5
- Le client JavaScript : soap.js
- L'interface utilisateur, elle aussi modifiée

MOTS-CLÉS

- Header, enveloppe SOAP
- Types XSD
- Persistance d'objets

► <http://en.wikipedia.org/wiki/DBus>

► <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>

Les protocoles de communication non-standards sont voués à l'échec. Les tentatives d'unification se multiplient : elles vont des projets Unix comme DBus, un protocole unifiant les communications entre programmes, à des projets orientés web, comme SOAP, qui standardise l'échange d'informations via le Web.

Si vous voulez accéder à la météo de votre région par exemple et l'afficher sur votre page personnelle, vous aurez besoin d'un service web. Votre langage de génération de pages (PHP, ASP, Perl...) devra se connecter au site qui propose le service météo et se faire comprendre pour obtenir la région demandée. On comprend alors tout l'intérêt des standards : l'enjeu consiste à faire communiquer des applications différentes, des langages différents, des environnements différents... d'une manière codifiée, dans un standard rigoureux, permettant les échanges, sans se noyer dans une jungle de protocoles incompatibles.

SOAP est l'une des réponses proposées : il signifie Standard Object Access Protocol. Il est le successeur de XML-RPC (XML-Remote Procedure Call), au départ protocole d'appel de méthodes distantes, très simple, encodé en XML, créé en partie par Microsoft. XML-RPC a été retravaillé et a évolué radicalement pour devenir un nouveau protocole : SOAP. Au départ soutenu par Microsoft, SOAP est maintenant pris en charge par le W3C. La dernière version est SOAP 1.2. Une requête SOAP typique consiste en un fichier XML (voir remarque) envoyé par méthode POST sur une page, qui interprète les données et renvoie dans le contenu une réponse, toujours au format XML. Dans notre cas, le client sera Mozilla et le serveur, PHP.

OUTILS Configurer PHP

Il devient maintenant nécessaire de configurer nous-mêmes PHP. Si vous gérez vous-même la configuration de votre serveur, ce sera aisé. Sous Unix, vous compilerez les sources avec l'option `--enable-soap`. Sous Windows, vous devrez enlever le « ; » devant la ligne `extension=php_soap.dll` pour activer celle-ci dans votre `php.ini`. Ce sera la même procédure si vous utilisez des paquets tout prêts, comme WAMP pour Windows.

► <http://www.wampserver.com>

Si par contre vous utilisez le mécanisme de votre distribution (sous Unix) pour avoir PHP, il faut vous référer aux manuels d'installation. Assurez-vous de bien disposer de la version 5 de PHP. Par exemple, la distribution Debian, à l'heure de la rédaction de ce livre, ne dispose toujours pas de PHP5. Vous devrez donc peut-être utiliser des sources alternatives, comme <http://www.dotdeb.org>. Installez ensuite l'extension SOAP, compilée en *shared object* pour permettre une installation séparée. Pour la distribution Gentoo, il faut « démasquer » les *ebuilds* de PHP5, encore marqués comme version instable (*unstable*), puis activer SOAP dans la variable USE.

Nous utilisons toujours la version 5 de PHP et cette fois ci, c'est au travers de l'extension SOAP intégrée que sera mis en valeur le modèle objet. Malgré quelques bogues dans sa jeunesse, cette extension devient très stable, surtout dans la version 5.1 de PHP. Cependant, elle n'est pas présente par défaut, il faudra donc penser à l'activer avant de commencer les tests !

SOAP en détail : application à XUL Forum

Concrètement, quelle va être la logique des communications SOAP avec XUL Forum ?

Le premier contact avec SOAP se fera lorsque l'utilisateur demandera à voir un message. En utilisant l'ID de ce message, qui est sous la forme `http://www.xulforum.org/sujets/[[id du sujet]]` éventuellement suivi de `/messages/[[id du message]]`, on pourra faire appel à la méthode `obtenirMessage(uri du message)` exposée par le serveur SOAP. Elle retournera bien sûr les différents constituants du message, qui seront ensuite traités par JavaScript.

ALTERNATIVES Comment se passer de SOAP ?

Nous choisissons d'illustrer la communication du client vers le serveur via SOAP. Il existe cependant d'autres méthodes.

- La plus simple est sans doute l'utilisation de la méthode GET : en appelant, avec un `XMLHttpRequest`, l'URL `http://www.example.com/monserveur.php?letitredumessage=XXX&lecontenudumessage=XXX` on peut passer au serveur des variables, récupérables grâce au tableau `$_GET` en PHP par exemple. Inconvénient : une URL ne peut pas dépasser 255 caractères.
- Une méthode POST : on recrée de toutes pièces la requête, en simulant l'envoi d'un formulaire HTML. Il faut bien sûr se plonger dans les normes RFC qui définissent la manière d'encoder un formulaire et une fois la requête construite, on l'envoie à l'aide de `XMLHttpRequest` (mais cette fois-ci en mode POST, un mode que nous n'avons pas eu l'occasion d'élucider dans cet ouvrage). Le wiki XULFr vous permettra, encore un fois, d'approfondir cette méthode (utilisable par exemple pour une page d'inscription directement intégrée à l'extension : on pourrait ainsi recréer le formulaire utilisé sur le

site web et même envoyer un fichier !). L'inconvénient est cependant une très grosse utilisation des composants XPCOM dès lors qu'il s'agit d'envoyer un fichier. De plus, une bonne connaissance des protocoles HTTP est requise.

► <http://xulfr.org/wiki/ApplisWeb/Request>

- Toujours dans les services web, Mozilla inclut une API XML-RPC. Avec l'extension de PHP dédiée à cet effet, il serait possible de repenser ce chapitre en fonction de XML-RPC. Cependant et c'est là l'inconvénient majeur, ce ne sont que des composants XPCOM qui sont proposés (et non pas des objets JavaScript instanciables comme avec SOAP : `var appel = new SOAPCall;`)
- Encore et toujours, l'alternative AJAX. Mais, et nous ne le répéterons jamais assez, pourquoi s'encombrer d'une nouvelle méthode de communication alors que SOAP existe déjà ? Si vous êtes tenté, vous pourrez néanmoins essayer des bibliothèques dédiées comme OpenRico (pour des pages web cependant).

► <http://openrico.org/home.page>

L'enregistrement d'un message se fera de même : ce sera la fonction `enregistrerMessage(titre, texte, auteur, date, uri du sujet)`. L'URI du sujet sera nul si c'est un nouveau sujet ; dans le cas contraire, il contiendra l'URI du sujet auquel répond le message.

Nous aurons aussi besoin d'une identification SOAP. En effet, pour des questions de sécurité, nous devons vérifier l'identité d'un utilisateur avant d'enregistrer des messages sous son nom dans la base de données. En effet, un autre client SOAP malveillant pourrait, si nous ne demandions pas l'identité du posteur, enregistrer des messages sous un faux nom.

Une première réponse à ce problème serait, lors d'appel à des fonctions nécessitant une identification, comme enregistrer un message, de fournir à la fonction le nom d'utilisateur et le mot de passe déjà donnés au départ. Mais alors un nouveau problème se pose : comment transmettre le nom d'utilisateur et le mot de passe, de `xulforum.xul` à `index.xul` ? Via l'URL ? Solution peu élégante... Redemander à l'utilisateur de s'identifier sur la seconde page ? Ce serait contraire à la logique de XUL Forum qui veut que toute l'identification se passe dans `xulforum.xul`.

Nous allons donc utiliser une fonctionnalité fort utile de PHP5 : la persistance des objets.

CULTURE À quoi ressemble un appel SOAP ?

Un appel SOAP utilise plusieurs espaces de nommage : XSD pour définir les différents types (chaîne, entier) selon XML, SOAP-ENC pour l'encodage d'une requête SOAP, SOAP-ENV pour les éléments structurels de la requête et NS1, qui correspond en fait à l'URI du serveur.

Ceci est une requête créée automatiquement par un client PHP pour l'appel de la méthode `identification`, avec comme paramètres le DN et le mot de passe de l'utilisateur.

On retrouve une enveloppe SOAP, le corps du message et, dedans, la requête pour la méthode `identification` (le nom NS1 est engendré automatiquement, de même que les noms des paramètres). Les deux paramètres sont de types chaîne (précisé grâce à XSD) et leur valeur est bien sûr encodée en Unicode (toutes les requêtes SOAP se feront en Unicode dans ce chapitre, encodage par défaut des deux parties, Mozilla et PHP).

```
<SOAP-ENV:Envelope SOAP-ENV:
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1="http://www.xulforum.org/xfServeur"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
  <SOAP-ENV:Body>
    <ns1:identification>
      <param0 xsi:type="xsd:string">
        uid=jonathan,ou=utilisateurs,o=xulforum
      </param0>
      <param1 xsi:type="xsd:string">*****</param1>
    </ns1:identification>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

L'idée est la suivante : du côté de PHP, on crée un objet caractéristique d'une session SOAP ; il sera gardé en l'état durant tout une même session. Du côté de Mozilla, d'abord on s'identifie et, sous réserve de succès, une propriété de l'objet PHP est marquée comme indiquant que l'utilisateur a bel et bien été identifié. Des fonctions utilitaires permettront de récupérer les informations caractéristiques de la session serveur : elles seront transmises à `index.xul` via l'URL (ou via les cookies de Mozilla). Ensuite, au prochain appel, sur `index.xul`, par exemple pour poster un message, le serveur « verra » que l'objet, qui a été gardé tel quel, a sa propriété `identOk` marquée à `true`. L'utilisateur s'est donc identifié par le passé : on peut enregistrer le message sous son nom sans crainte.

ALTERNATIVES **La vie sans PHP5 et son extension SOAP**

Pour ceux qui ne peuvent pas ou ne veulent pas utiliser l'extension livrée avec PHP5, il existe une classe `PEAR::SOAP`, ou même une classe `NuSOAP` qui permettent toutes deux de créer des clients ou serveurs SOAP. Ce sont les plus connues.

- ▶ <http://pear.php.net/package/SOAP>
- ▶ <http://dietrich.ganx4.com/nusoap/>

Le serveur en pratique avec PHP5

Il est temps de mettre en pratique ce fonctionnement théorique : ce sera le fichier `soap.php`.

Le serveur, vue globale

```
<?php
error_reporting(E_STRICT);
include("../global.inc.php");
class xfServeur {

    private $dejaVu = 0;
    private $identOk = 0;

    private $db;
    private $ldapHost, $mysqlHost, $mysqlUser, $mysqlPass;

    function __construct($pLdapHost, $pMysqlHost, $pMysqlUser,
        $pMysqlPass) {
        /* initialisation de tout ce qui concerne les sessions */
        session_start();
        /* pour mysql */
        $this->mysqlHost = $pMysqlHost;
        $this->mysqlUser = $pMysqlUser;
        $this->mysqlPass = $pMysqlPass;
        /* pour ldap */
        $this->ldapHost = $pLdapHost;
    }
}
```

- ◀ Pour plus de rigueur, nous voulons voir toutes les erreurs. Pour plus de clarté, les informations de connexion sont placées dans un fichier à part.
- ◀ La classe, une fois instanciée, formera un objet qui sera conservé durant toute la session SOAP.
- ◀ Deux variables utilitaires. La première permet de déterminer si c'est la première fois que l'objet est créé dans la session (valeur 0 par défaut, sinon valeur 1). La seconde permet de déterminer si une identification réussie a déjà eu lieu dans cette session ou non.
- ◀ Les variables membres sont conservées mais pas les ressources, qui durent le temps d'un script. Il faudra donc à chaque fois recréer la connexion à la base de données (variable `db`) et pouvoir connaître les informations de configuration.
- ◀ Le constructeur est appelé à chaque fois que l'on instancie l'objet. On démarre la session, on passe les informations servant à la connexion dans les variables membres et l'on est prêt pour servir les différentes requêtes !

L'identification sera détaillée plus loin : elle accepte bien sûr le nom (le DN) et le mot de passe de l'utilisateur.

Cette autre fonction utilitaire renvoie des informations sur la session : elle permettra, entre autres, de savoir si la session dont le client a les références n'est pas « périmée ».

L'unique paramètre est l'URI du message voulu, par exemple `http://www.xulforum.org/sujets/1/`

Pour enregistrer un message. Il y aura bien sûr une vérification préalable sur `$this->identOk` avant de procéder à l'enregistrement du message.

Une autre fonction utilitaire, qui permet d'établir la connexion à la base de données avant toute opération sur cette dernière. La classe `SOAPFault` est très pratique pour envoyer simplement des messages d'erreur SOAP. Notez bien que le fichier PHP est édité au format Unicode, pour que les accents dans les erreurs SOAP soient eux aussi en Unicode. Pour la deuxième erreur, c'est le message d'erreur MySQL, plus parlant, qui est fourni.

Une dernière fonction utilitaire : avant de retourner les résultats de la base de données (qui elle est en latin1), il faut les encoder en Unicode, par l'intermédiaire de la fonction PHP `utf8_encode`.

Une sous-fonction : lorsqu'on demande à visualiser un message, ce sera soit un sujet,...

... soit un message. Les fonctions adaptent en fait la requête MySQL à utiliser.

La classe est fermée.

Maintenant, voici les instructions exécutées à chaque lancement du script. On crée un nouveau serveur (le premier paramètre sera explicité plus loin dans ce chapitre), avec l'URI l'identifiant. On lui associe la classe `xfServeur`, dont les méthodes publiques deviennent des méthodes du serveur SOAP (les paramètres passés après le nom de la classe sont transmis au constructeur). On indique aussi que l'objet `xfServeur` doit persister durant toute la session.

```
public function identification($pNom, $pPass) {
}

public function infosSession() {
    return array(session_name(), session_id(), $this->identOk);
}

public function obtenirMessage($pMsg) {
}

public function enregistrerMessage($titre, $texte, $auteur,
    $date, $sId) {
}

private function _connecterMySQL() {
    if ($this->db != null) return;
    $this->db = @mysql_connect($this->mysqlHost,
        $this->mysqlUser, $this->mysqlPass);
    if (!$this->db)
        throw new SOAPFault("SOAP:Server/SQL",
            "Impossible de se connecter à la base de données");
    $r = @mysql_query("USE jonathan", $this->db);
    if (!$r)
        throw new SOAPFault("SOAP:Server/SQL",
            mysql_error($this->db));
}

private function _toutEnUnicode($o) {
    foreach ($o as $p => $v) {
        $o[$p] = utf8_encode($v);
    }
    return $o;
}

private function _chercherSujet($sId) {
}

private function _chercherMessage($sId, $mId) {
}

}

$s = new SOAPServer(null, array('uri' =>
    'http://www.xulforum.org/xfServeur'));
$s->setClass("xfServeur", $ldapHost, $mysqlHost, $mysqlUser,
    $mysqlPass);
$s->setPersistence(SOAP_PERSISTENCE_SESSION);
```

```

if ($_SERVER['REQUEST_METHOD'] == "POST") {
    $s->handle();
}
else {
    header("Content-type: text/plain");
    echo "Fonctions supportées par ce serveur SOAP : \n";
    foreach($s->getFunctions() as $f)
        echo $f."\n";
}
?>

```

- ◀ SOAP procède par méthode POST. Si l'on a à faire à une requête de ce type, le serveur doit entrer en action.
- ◀ Si ce n'est pas le cas, on envoie en texte brut la liste des fonctions proposées par le serveur.
- ◀ Fin du fichier PHP.

La simplicité est redoutable : il n'y a pas besoin de tenir à jour une liste des fonctions du serveur SOAP, des paramètres qu'elles acceptent, qu'elles retournent... il suffit d'écrire les méthodes de la classe et de déclarer celles-ci comme publiques. Cette simplicité a cependant une contrepartie, que nous verrons au moment d'analyser les résultats avec JavaScript.

Permettre l'authentification de l'utilisateur

La méthode d'identification est indispensable : si vous avez bien compris la logique précédente, elle ne présente aucune difficulté.

```

public function identification($pNom, $pPass) {
    if ($pNom == null && $pPass == null) return 0; ❶
    if ($this->dejaVu && $this->identOk) {
        return 1; ❷
    } else {
        $l = @ldap_connect($this->ldapHost);
        if (!$l)
            throw new SoapFault("SOAP:Server",
                "Impossible de se connecter à LDAP");
        @ldap_set_option($l, LDAP_OPT_PROTOCOL_VERSION, 3);
        $b = @ldap_bind($l, $pNom, $pPass); ❸
        ldap_close($l);

        $this->dejaVu = 1; ❹
        if ($b) {
            $this->identOk = 1;
            return 1; ❺
        } else {
            $this->identOk = 0;
            return 0; ❻
        }
    }
}

```

Si les deux paramètres passés sont vides ❶, on retourne faux, car on ne veut pas d'une identification anonyme. De même, si l'utilisateur a déjà utilisé cet objet et qu'une identification précédente a réussi ❷, on retourne vrai.

ALTERNATIVES addFunction()

On peut se passer de la notion de classe et ajouter ses fonctions via la méthode `addFunction`, qui accepte le nom d'une fonction, un tableau de noms de fonctions, ou la constante `SOAP_FUNCTION_ALL`. Cependant, on perd la persistance et, de plus, l'utilisation de la constante `SOAP_FUNCTION_ALL` enregistre toutes les fonctions de PHP disponibles ! À déconseiller bien sûr...

Si aucun de ces deux cas n'est vérifié, il faut procéder à une identification LDAP. On se connecte, on utilise la version 3 du protocole (qu'OpenLDAP utilise) et on garde les résultats de l'identification ❸ dans la variable `b`.

Si l'identification a réussi, on modifie la variable `identOk` en conséquence et l'on retourne vrai ❺, sinon, on retourne faux ❻. Dans les deux cas, on enregistre le passage de l'utilisateur ❹ : il pourra servir par exemple à des statistiques, comme le nombre d'appels SOAP déjà effectués... (on utilisera alors `$this->dejaVu++`);).

Lire un message

Une fois sur `index.xul` et correctement identifié, l'utilisateur doit être en mesure de récupérer un message : c'est la fonction `obtenirMessage()`.

```
public function obtenirMessage($pMsg) {
    $parties = explode("/", $pMsg); ❶
    if (count($parties) == 4 || count($parties) == 5) { ❷
        //pour prendre en compte le / final
        $sId = $parties[4];
        $sujet = $this->_chercherSujet($sId); ❸
        return $sujet;
    } else if (count($parties) == 6 || count($parties) == 7) { ❹
        $sId = $parties[4];
        $mId = $parties[6];
        $message = $this->_chercherMessage($sId, $mId); ❺
        return $message;
    } else { ❻
        throw new SOAPFault("SOAP:Serveur",
            "URI de message ou sujet demandé inconnue");
    }
}
```

Le principe est très simple : on sépare les constituants de l'URI ❶ selon le séparateur « / », via la fonction `explode`.

```
jonathan@daisy ~ $ php -a
Interactive mode enabled

<?php
print_r(explode("/", "http://www.xulforum.org/sujets/0/"));
Array
(
    [0] => http:
    [1] =>
    [2] => www.xulforum.org
    [3] => sujets
    [4] => 0
    [5] =>
)
```

Si la longueur correspond à celle d'un sujet ❷, on appelle la méthode utilitaire `_chercherSujet()` ❸. Si la longueur correspond à celle d'un message ❹, c'est ce message en particulier que l'on va chercher via `_chercherMessage()` ❺. Si la longueur n'est pas reconnue, on lance une erreur ❻.

Nous ne détaillerons que la méthode `_chercherMessage()`, car `_chercherSujet()` est exactement similaire.

```
private function _chercherMessage($sId, $mId) {
    $this->_connecterMySQL();
    $r = @mysql_query("SELECT titre,texte,auteur,date
        ➤ FROM xf_messages WHERE s_id='$sId' AND id='$mId'",
        $this->db);
    if (!$r)
        throw new SOAPFault("SOAP:Server/SQL",
            mysql_error($this->db));
    return $this->_toutEnUnicode(@mysql_fetch_row($r));
}
```

La seule chose à remarquer ici est le type de retour : c'est un tableau ne contenant que des index (fonction `mysql_fetch_row` ; `mysql_fetch_array` renvoie un tableau associatif avec en clés les noms des champs et `mysql_fetch_object` un objet). L'index 0 contiendra le titre, l'index 1 le texte et ainsi de suite, dans l'ordre des champs demandés via la requête MySQL. On spécifie deux conditions, que l'ID du sujet et que l'ID du message soient tous les deux bons. Ceci rajoute une sécurité supplémentaire quant à l'exactitude des relations sujet / message.

Enregistrer un message

Enfin, dernière méthode, très utile, celle qui permet d'enregistrer un message.

```
public function enregistrerMessage($titre, $texte, $auteur,
    $date, $sId) {
    if (!$this->identOk)
        throw new SOAPFault("SOAP-ENV:Server",
            "Pas identifié, veuillez réessayer !"); ❶
    $this->_connecterMySQL();
    $texte = strip_tags($texte,
        "<b> <u> <i> <a> <ul> <li> <ol>");
    $titre = addslashes(utf8_decode($titre));
    $texte = addslashes(utf8_decode($texte));
    $texte = nl2br($texte); ❷
    $auteur = addslashes(utf8_decode($auteur));
    if (!$sId) { ❸
        $r = @mysql_query(
            "INSERT INTO xf_sujets(titre, texte, auteur, date)
            ➤ VALUES('$titre', '$texte', '$auteur', NOW())");
```

ASTUCE

Conventions de codage, underscore

Vous l'aurez remarqué, tout ce qui est privé est préfixé par un signe `_` (underscore) dans ce chapitre. C'est en fait une convention de codage parfois utilisée en C, permettant de distinguer les fonctions qui sont utilisées seulement au sein d'un fichier et celles qui sont utilisables par les autres fichiers.

Ainsi, dans `soap.js`, toutes les routines qui ne sortiront pas du cadre de ce même fichier commenceront par `_`, tandis que les fonctions appelées depuis les scripts principaux, `forum.js` ou `identification.js`, ne posséderont pas ce caractère underscore.

```

if (!$r)
    throw new SOAPFault("Server:SQL", mysql_error());
} else { ❷
    $parties = explode("/", $sId);
    $sId = $parties[4];
    $r = @mysql_query("INSERT INTO xf_messages(
        ➤ s_id, titre, texte, auteur, date) VALUES
        ➤ ('$sId', '$titre', '$texte', '$auteur', NOW())");
    if (!$r) ❸
        throw new SOAPFault("Server:SQL", mysql_error());
    }
}

```

Tout d'abord, si on n'est pas identifié ❶, le serveur refuse d'enregistrer le message. Ensuite, on applique quelques transformations aux paramètres fournis ❷ : le HTML est autorisé, mais seulement pour une liste de balises réduite. La base de données est en latin-1. On ne doit pas lui donner des informations en Unicode : on décode donc l'UTF8. Enfin, les sauts de lignes doivent être convertis en balises `
` : c'est la fonction `n12br`, « new line to br ». C'est en effet du HTML que nous stockons dans la base de données. Si ce message n'est pas en réponse à un sujet particulier ❸, c'est que c'est un nouveau sujet. On insère donc une nouvelle valeur dans la table `xf_sujets` et la date courante est obtenue grâce à la fonction MySQL `NOW()` qui permet de remplir des champs `DATE` et `DATETIME`.

ASTUCE Tester souvent, par petits morceaux

Ces fonctions n'ont pas encore été mises en relation avec l'interface XUL : vous pouvez donc et c'est une bonne pratique, effectuer lors de vos développements ce que l'on appelle des tests unitaires. L'idéal est d'écrire en premier une suite de fonctions décrivant le comportement attendu du fichier `soap.js`.

```

initialiserSoap("jonathan");
dump(identifieurSoap("uid=jonathan,ou=utilisateurs,o=xulforum", "*****")+ "\n");
//on attend vrai
var r = verifierSessionSoap();
if (r)
    dump(gPhpSessName+"="+gPhpSessId+" "+r[2]+ "\n"); //on attend PHPSESSID=XXX suivi de 1
obtenirMessageSoap("http://www.xulforum.org/sujets/2/messages/3/");
//on peut aussi mettre des dump() directement dans obtenirMessageSoap
posterMessageSoap("Titre d'un message", "Un texte", "http://www.xulforum.org/sujets/1/");
//vérifier dans la base de données si ça a bien marché

```

Ainsi, on verra au fur et à mesure si les fonctions adoptent bien le comportement attendu et dans le cas d'une erreur, on aura de suite le test permettant de la détecter, test facilement modifiable. Si l'on avait de suite testé `soap.js` depuis les fichiers XUL, les

ainsi, en réfléchissant aux valeurs de retour, aux paramètres acceptés, vous pourriez avoir une vision claire des fonctions SOAP, plus efficace qu'un tâtonnement exigeant de nombreux remaniements ultérieurs. Par exemple :

erreurs auraient été beaucoup plus difficiles à détecter : la succession des fonctions aurait été éclatée sur plusieurs autres procédures, les informations affichées sur la console auraient été moins faciles à identifier, au milieu d'autres messages...

Si ce message est en réponse à un sujet particulier ④, on récupère l'identifiant du sujet au milieu de son URI et on crée la bonne requête, pour la table `xf_messages` cette fois-ci. Enfin, s'il y a eu une erreur, on en informe le client par l'intermédiaire d'une faute SOAP ⑤. Tout ce traitement PHP est en fait assez « classique » : il faut juste bien comprendre le mécanisme des objets persistants et bien implémenter les requêtes et l'identification. Le gros du travail va se faire ensuite, ce sera dans Mozilla.

Le client JavaScript

Ceci sera le dernier fichier JavaScript que nous écrirons dans `content` : il s'appellera `soap.js` et contiendra toutes les routines de traitement SOAP. Les différentes parties seront en fait des échos aux méthodes du serveur PHP : il y a des correspondances entre les deux. Nous allons voir comment procéder à l'identification, au rapatriement d'un message et à l'enregistrement d'un message ou d'un sujet.

L'initialisation

De même qu'au chapitre 10 nous avons initialisé LDAP, nous allons ici aussi offrir une fonction d'initialisation SOAP.

```
var gPhpSessName;
var gPhpSessId;
var gAuteur;

function initialiserSoap(pAuteur, pPhpSessName, pPhpSessId) {
    netscape.security.PrivilegeManager.enablePrivilege(
        "UniversalBrowserRead");

    gAuteur = pAuteur;
    if (arguments.length == 3) {
        gPhpSessName = pPhpSessName;
        gPhpSessId = pPhpSessId;
        verifierSessionSoap();
    } else {
        var r = verifierSessionSoap();
        gPhpSessName = r[0];
        gPhpSessId = r[1];
    }
}
```

Cette fonction sera appelée avant tout traitement SOAP. Elle prend un nombre variable d'arguments : s'il y en a trois, cela signifie que c'est la page `index.xul` qui initialise SOAP, car elle possède déjà les identifiants de session, récupérés sur `xulforum.xul`. On place donc ces identifiants de

session dans des variables globales et on vérifie que la session est toujours valide. Si l'on est sur `xulforum.xul`, on n'a pas encore d'identifiant de session : il faut donc en obtenir un et le garder via les variables globales.

La première ligne, héritage du code original de Netscape, indique que le navigateur a besoin de certains privilèges pour que les objets JavaScript ne soient pas gênés dans leurs actions de rapatriement ou d'envoi d'informations depuis ou sur le serveur. Les composants XPCOM n'ont pas ce problème car ils ne sont autorisés que pour les extensions, on est donc en terrain autorisé si l'on fait appel à une méthode d'un composant XPCOM.

Est-on en phase avec le serveur ? Vérification de la session

Il faut d'abord vérifier si la fonction n'a pas retourné faux avant d'accéder aux différents index du tableau de retour.

```
function verifierSessionSoap() {
    var retour = _appelSoap("infosSession", {}); ❶
    if (!retour) return false; ❷

    var valeurs = String(retour[0].value).split(","); ❸
    if (valeurs[1] != gPhpSessId) { ❹
        gPhpSessName = valeurs[0];
        gPhpSessId = valeurs[1];
    }
    valeurs[2] = Number(valeurs[2]); ❺
    return valeurs; ❻
}
```

La fonction permettant de vérifier les informations de session se contente d'appeler ❶, grâce à une routine interne, la méthode `infosSession` du serveur. Elle vérifie que cet appel n'a pas échoué ❷ et effectue ensuite un traitement un peu particulier.

Normalement, le serveur PHP renvoie un tableau indexé. En pratique, c'est une liste de valeurs séparées par des virgules qui arrive en JavaScript. Pour rétablir le tableau original, il faut séparer cette chaîne en suivant le délimiteur « , » ❸. La variable `valeurs` contient alors le tableau qui a été réellement envoyé par PHP. Si ce sont des informations de session différentes de ce que nous avons déjà, nous mettons à jour nos variables globales ❹.

Dernier point, il faut transformer la chaîne correspondant à la variable `identOk` en booléen ❺, car nous allons effectuer des tests conditionnels dessus. En effet, une chaîne comme "0" renvoie « vrai » dans un test, car ce n'est pas une chaîne vide. Finalement, on retourne le tableau contenant les informations de session ❻.

Pas de duplication de code : une routine pour tous nos appels SOAP

```
function _transportUri() {
    return 'http://' +
        gConfig.php.hote + gConfig.php.chemin + 'soap.php?' +
        gPhpSessName + "=" + gPhpSessId;
}

function _appelSoap(methode, parametres) {
    document.getElementById("xf-statusbar-progres").
        ➔ setAttribute("mode", "undetermined"); ❶

    var appel = new SOAPCall(); ❷
    appel.transportURI = _transportUri(); ❸
    var p = new Array();
    var i = 0;
    for (a in parametres)
        p[i++] = new SOAPParameter(parametres[a], a);
    appel.encode(0, methode, "http://www.xulforum.org/xfServeur",
        0, null, p.length, p); ❹

    var r = appel.invoke(); ❺
    document.getElementById("xf-statusbar-progres").
        ➔ setAttribute("mode", "determined"); ❻

    if (r.fault) { //nsISoapFault ❼
        ajouterErreur(gStringBundle.getFormattedString(
            "erreurSoap", new Array(String(r.fault.faultCode),
            String(r.fault.faultString))));
        return false;
    } else {
        return r.getParameters(false, {}); ❽
    }
}
```

C'est dans cette routine que nous manipulons vraiment les objets SOAP. Il faut tout d'abord préciser que tout se fait en mode synchrone : la version asynchrone est décrite en remarque. On place donc le progress-meter en mode indéterminé, pour indiquer qu'une action est en cours ❶. Ensuite, on instancie un objet `SOAPCall`, à la base des fonctions SOAP de Mozilla ❷. On lui indique l'adresse de la page serveur à laquelle il doit envoyer les informations ❸, puis on procède à un traitement permettant de transformer un objet de type `{ "nomduparamètre" : valeurduparamètre }` en tableau d'objets `SOAPParameter`. Ceci permet de masquer les sous-traitements SOAP et d'autoriser des appels simplifiés depuis les autres fonctions de `soap.js`, car la syntaxe objet est plus modulable et plus simple que la construction d'un tableau d'objets SOAP. Vous remarquerez que lorsqu'on encode la requête, on précise aussi l'URI du serveur, déjà rentré dans le serveur PHP ❹.

Une fois que tout est prêt, on peut lancer la requête : on l'« invoque » ⑤. Lorsqu'elle est terminée, on arrête le défilement ⑥.

Il faut maintenant vérifier ce que l'appel nous a renvoyé. Si c'est un booléen faux ⑦, il y a eu erreur et on le signale à l'utilisateur. Si tout est bon, on demande à obtenir les paramètres de retour, via la méthode `getParameters()`. Bien qu'envoyés par PHP, ils sont, eux aussi, des instances de l'objet `SOAPParameter` ; on les renvoie à la fonction appelante via l'instruction `return` ⑧.

La valeur d'un paramètre SOAP est accessible via sa propriété `value` : c'est ce qui a été fait plus haut, lorsqu'il s'agissait d'obtenir des informations sur une session.

ALTERNATIVE Appels SOAP asynchrones

Au lieu d'utiliser `SOAPCall::invoke()`, il est possible d'utiliser son équivalent asynchrone : `SOAPCall::aSyncInvoke()`. Son unique argument est une fonction, à laquelle seront transmis trois paramètres :

- l'objet `SOAPResponse` retourné ;
- l'objet `SOAPCall` à l'origine de l'appel ;
- un code de retour : 0 indique le succès, une valeur autre indique un échec.

On pourra donc écrire :

```
function callback(r, c, e) {
    //traitement...
}
```

```
appel.aSyncInvoke(callback);
```

Les méthodes asynchrones sont généralement recommandées, toujours pour la même raison : ne pas geler l'affichage.

CULTURE Les différents paramètres de `invoke()` et `getParameters()` expliqués

La méthode `invoke()` accepte de nombreux paramètres :

- La version SOAP, déterminée par une constante.
 - <http://xulplanet.com/references/objref/SOAPCall.html>
- Le nom de la méthode et l'URI du serveur appelé, tous deux vus dans notre cas.
- Le nombre de blocs d'en-tête (header).
- Un tableau similaire à celui utilisé pour les paramètres, mais avec des objets `SOAPHeaderBlock` cette fois-ci. Les en-têtes peuvent être utilisés pour passer des paramètres comme une langue préférée, un format d'image demandé, à la manière des en-têtes HTTP.
- Le nombre de paramètres et le tableau les contenant, tous deux vus dans notre exemple.

La méthode `getParameters()` accepte quant à elle deux arguments :

- Le premier est un booléen. S'il est vrai, c'est un document DOM qui est retourné, dont l'analyse pourra être faite pour extraire nous-mêmes les paramètres. S'il est faux, l'extension SOAP doit elle-même tenter de récupérer les paramètres et les analyser (d'où les incorrections rencontrées avec les tableaux retournés par PHP). En mettant sa valeur à vrai, on pourra tenter de recréer les types de PHP, comme les tableaux associatifs ou les objets, en faisant une analyse DOM.
 - <http://www.onlamp.com/pub/a/onlamp/2005/06/23/mozsoap.html>.
- Le second paramètre est un objet, amené à contenir le nombre de paramètres reçus. Nous n'en avons pas l'utilité car nous savons le nombre de paramètres retournés.

Le moment crucial : l'identification

```
function identifierSoap(pUtilisateur, pPass) {
    var retour = _appelSoap("identification",
        { "param0" : pUtilisateur, "param1" : pPass });
    if (!retour) return false;
    var valeur = Number(retour[0].value);

    if (valeur == 1) {
        dump("Identification réussie pour "+pUtilisateur+
            "/" +pPass+"\n");
        return true;
    } else {
        ajouterErreur(gStringBundle.getString("identificationEchec"));
        return false;
    }
}
```

La procédure d'identification ne fait que réutiliser ce qui a été vu précédemment : on appelle la méthode du serveur avec les paramètres (dont le nom importe peu puisque ni PHP, ni JavaScript ne s'en soucient) requis : DN et mot de passe de l'utilisateur. On vérifie le résultat qui nous est fourni, on le transforme en nombre et on agit en conséquence : valeur de retour true si succès, erreur et return false en cas d'échec.

Indispensable : lire un message

```
function obtenirMessageSoap(pUri) {
    var r = _appelSoap("obtenirMessage", { "param1" : pUri });
    if (!r)
        return false;
    var o = r[0].value.slice(",");
    return {"titre" : o[0], "texte" : o[1], "auteur" : o[2],
        "date" : o[3] };
}
```

Il n'y a ici qu'une variante : on retourne un objet plutôt qu'un tableau, pour plus de clarté dans les traitements futurs.

Vital : poster nos propres messages

```
function posterMessageSoap(pTitre, pTexte, pSId) {
    var v = verifierSessionSoap();
    if (!v) {
        ajouterErreur(gStringBundle.GetString("pasIdentifie"));
        return false;
    }
    if (!v[2]) {
        ajouterErreur(gStringBundle.GetString("pasIdentifie"));
        return false;
    }
    var r = _appelSoap("enregistrerMessage", {
        "param0" : pTitre,
        "param1" : pTexte,
        "param2" : gAuteur,
        "param3" : null,
        "param4" : pSId
    });
    return r;
}
```

Cette dernière fonction nécessite plus de vérifications :

- d'abord, est-ce que le rapatriement des infos session a réussi ?
- si oui, sommes-nous identifié ?

Si l'une de ces deux conditions n'est pas satisfaite, on arrête immédiatement la fonction. Dans le cas contraire, on appelle la fonction serveur et on renvoie la variable `r`, représentant le succès de l'opération « enregistrement du message ».

POUR ALLER PLUS LOIN **Autres fonctions**

Il y a bien sûr les autres fonctions à implémenter : édition d'un sujet et suppression, nombre de messages au total, nombre de messages non lus... Pour les deux premières, une double vérification sera nécessaire, côté Mozilla et côté serveur. Bien sûr, si le nom d'utilisateur est « jonathan », la suppression et la modification seront possibles pour tous les messages...

Plus sérieusement, un vrai système d'identification consisterait à définir un champ spécial dans l'annuaire LDAP, qui indiquerait si la personne est ou non administrateur. Une vérification rapide sur ce champ au moment de permettre la modification / suppression d'un message serait une bonne manière de rendre compte du statut « administrateurs du forum ».

Les modifications de l'interface avec XUL

En fait, la majeure partie des modifications va se concentrer dans le widget XBL vu précédemment. Nous allons rajouter un bouton *R* accolé à la croix permettant de quitter et, suite à un double-clic sur ce *R*, le widget se transformera en champ de formulaire pour saisir un message.

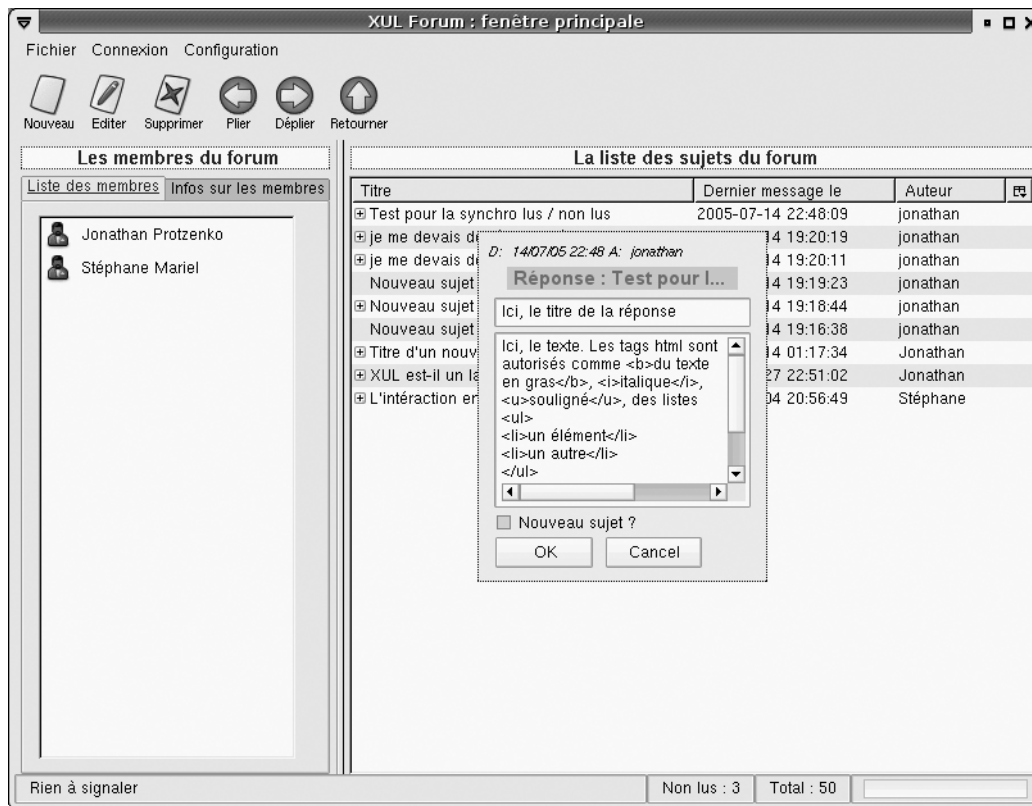


Figure 12-1 La fenêtre de saisie d'une réponse

Changements dans le fichier XBL

Les manipulations DOM sont très similaires à celles que nous avons vues dans le chapitre précédent : l'astuce qui consiste à utiliser la classe d'un élément au lieu de son ID pour l'identifier est toujours valable. Car il faut bien garder à l'esprit que dans un document XML, un ID est unique. Or il y a plusieurs widgets `fenetreMsg`, donc plusieurs cases « nouveau sujet » à cocher et ce pour le même document `index.xul`. Il est donc impossible d'utiliser l'attribut ID sur les enfants d'un widget XBL.

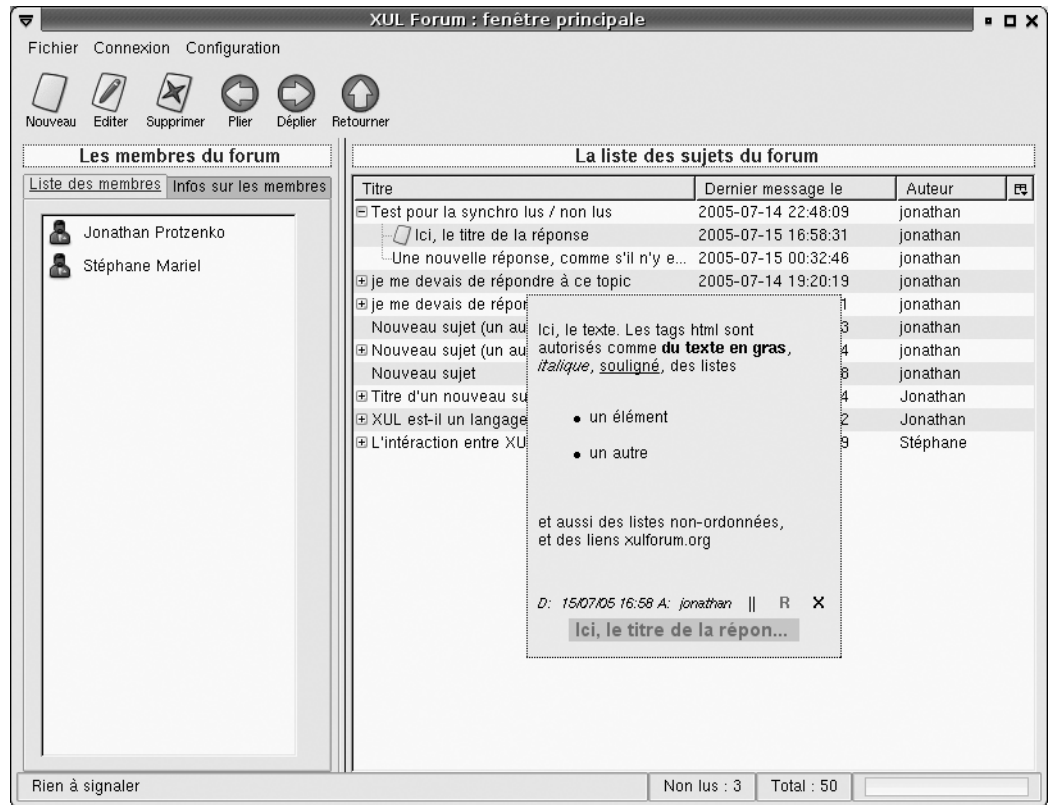


Figure 12–2 Le message qui en résulte

Nous ne détaillerons pas les manipulations DOM, consistant à transformer une fenêtre de visualisation d'un message en fenêtre de réponse et vice-versa : ce ne serait que du discours inutile. En revanche, la nature du formulaire est plus intéressante. Il est placé bien sûr dans la balise `<content>` du widget XBL.

Le formulaire XUL de réponse à un message, placé dans le widget XBL

```
<xul:vbox>
  <xul:textbox class="texteTitre" style="width: 200px;"/>
  <xul:textbox class="texteTexte" multiline="true" rows="8"
    style="width: 200px" />
  <xul:checkbox label="Nouveau sujet ?" checked="false"
    class="cocheSujet" />
  <xul:hbox>
    <xul:button label="OK" class="boutonOk" />
    <xul:button label="Cancel" class="boutonAnnuler" />
  </xul:hbox>
</xul:vbox>
```

Une largeur minimale est assignée aux deux champs texte. Le second est transformé en équivalent de `<html:textarea>` grâce à une hauteur de huit lignes et à l'attribut `multiligne`. Les classes CSS ne sont pas utilisées pour ces deux champs, elles servent uniquement à les identifier comme expliqué plus haut.

Suit une case à cocher permettant à l'utilisateur de spécifier qu'il s'agit d'un nouveau sujet qu'il veut poster et enfin, une boîte contenant les boutons *OK* et *Annuler*. On serait tenté de leur assigner un attribut `oncommand="this.posterReponse()"` ou `"this.annulerReponse()"`. Ce serait oublier que l'on est dans un widget XBL : il faut capturer l'événement `oncommand` dans les `<handlers>`.

Dans un widget XBL, il faut capturer les clics sur les boutons OK et Annuler

```
<handler event="command">
  switch (event.originalTarget.className) {
    case "boutonOk" :
      this.posterReponse();
      break;
    case "boutonAnnuler" :
      this.cacherReponse();
      break;
  }
</handler>
```

Des modifications mineures sont apportées aux fonctions `plier()` et `deplier()` : un attribut `<field name="enReponse">false</field>` fait son apparition et indique à ces deux fonctions si elles doivent montrer l'élément `<xul:vbox>` lors d'un dépliage, ou au contraire le masquer et montrer l'élément `<p>` qui contient le corps du message. La fonction `cacherReponse()` masque le formulaire (élément `<xul:vbox>`) et rétablit la visibilité de la réponse (élément `<p>`).

Maintenant (et c'est là le cœur de la procédure d'enregistrement d'un message), la fonction `posterReponse()` entre en jeu :

```
var c = document.getAnonymousNodes(this)[0].childNodes;
var c1;
for (var i = 0; i < c.length; ++i) { ❶
  if (c[i].nodeName == "xul:vbox") {
    c1 = c[i].childNodes;
    break;
  }
}
var titre, texte, nouveauSujet;
for (var i = 0; i < c1.length; ++i) { ❷
  dump(c1[i].nodeName+"\n");
  if (c1[i].nodeName[0] == "#") ❸
    continue;
```

```

    if (c1[i].className == "texteTitre") { ❷
        titre = c1[i].value;
        c1[i].value = null;
    }
    else if (c1[i].className == "texteTexte") { ❸
        texte = c1[i].value;
        c1[i].value = null;
    }
    else if (c1[i].className == "cocheSujet") { ❹
        nouveauSujet = c1[i].checked;
        c1[i].checked = false;
    }
}
if (titre == "" || texte == "") { ❺
    ajouterErreur(gStringBundle.getString("texteVide"));
    return;
}
if (nouveauSujet) ❻
    var r = posterMessageSoap(titre, texte, "");
else ❼
    var r = posterMessageSoap(titre, texte,
        this.getAttribute("sId")); ❽
if (!r) ❾
    ajouterErreur(gStringBundle.getString("messagePasEnvoye"));
else
    this.cacherReponse();

```

Comme on opère sur du contenu anonyme, on parcourt les enfants du premier bloc div ❶ : lorsqu'on tombe sur l'élément `xul:vbox`, conteneur pour notre formulaire, on quitte la boucle. On parcourt ensuite les différents éléments de cette même boîte verticale `xul:vbox` ❷ : si c'est un nœud `#text` ou `#comment`, on passe directement à l'itération suivante, via l'instruction `continue` ❸, évitant ainsi un accès à une propriété `className` qui provoquerait des erreurs. On tente ensuite de récupérer successivement le titre ❹, le texte ❺ du message et la valeur de la case cochée ❻.

Une fois ces opérations « d'extraction » de valeurs terminées, on vérifie que l'on n'envoie pas un message nul ❺ : si c'est le cas, on signale l'erreur et on arrête la fonction.

Si c'est un nouveau sujet qui est en fait posté ❽, on passe un URI de sujet vide, pour que le serveur PHP crée un nouveau sujet. Si c'est une réponse à un sujet ❹, on passe l'URI du message, contenu dans la propriété `sId` du widget XBL. On fait ensuite appel à la fonction SOAP dédiée ❽.

Enfin, un dernier test sur la valeur retournée ❾ permettra de s'assurer que l'enregistrement du message a bien eu lieu.

Changements dans les fichiers JavaScript

Nous avons vu l'utilisation de la méthode `postMessageSoap()`. Les autres méthodes de `soap.js` sont utilisées dans les deux fichiers `identification.js` et `forum.js`.

Sur `xulforum.xul`, au moment de la vérification de l'identité de l'utilisateur, nous écrirons :

```
if (!identificationSOAP())
    return;
identificationLDAP();
```

Tous les appels SOAP étant synchrones, il n'y a pas de problèmes de callback comme ceux rencontrés avec les appels LDAP. On peut donc attendre que l'identification SOAP se termine pour passer à la suite. La fonction `identificationSOAP()` est définie de la manière suivante :

```
var dn = "uid="+document.getElementById("xf-ident-ident_nom").value+", "+gConfig.ldap.baseDn;
var pass = document.getElementById("xf-ident-ident_pass").value;
initialiserSoap(document.getElementById("xf-ident-ident_nom").value);
identifierSoap(dn, pass);

var r = verifierSessionSoap();
gPhpSessName = r[0];
gPhpSessId = r[1];

if (r != false && r[2]) {
    return true;
} else {
    ajouterErreur("Erreur identification SOAP");
    return false;
}
```

Il faut d'abord initialiser SOAP, avec le nom d'utilisateur fourni et s'identifier selon les informations rentrées par l'utilisateur. La valeur de retour de `verifierSessionSoap()` permet d'obtenir les informations de session, stockées dans deux variables globales. Le dernier test permet de voir si l'identification a fonctionné ou pas.

Les deux variables de session vont être transmises par l'URL (sous réserve d'identification valide, bien sûr) :

```
var url = "chrome://xulforum/content/index.xul?config="+
    gConfig.url+"&phpsessname="+gPhpSessName+"&phpsessid="+
    gPhpSessId+"&auteur="+
    document.getElementById("xf-ident-ident_nom").value;
document.location.href=encodeURIComponent(url);
```


POUR ALLER PLUS LOIN

Analyseur dynamique d'URL

Un bon exercice consisterait à écrire une fonction qui analyse l'URL du fichier et crée directement un objet, sous la forme :

```
{ nomdelavariable : savaleur,
  nomdelasecondevariable : savaleur
}.
```

Ceci permettrait de s'affranchir d'une structure d'URL fixe. Si vous voulez, vous pouvez même rendre ces variables globales, grâce à la fonction :

```
eval("var "+nomdelavariable+" = "
+savaleur+");");
```

Pour être récupérées à la page suivante, index.xul :

```
var l = decodeURI(document.location.href);
var vars = l.split("&");
var phpSessName = vars[1].substr(vars[1].indexOf("=")+1);
var phpSessId = vars[2].substr(vars[2].indexOf("=")+1);
var auteur = vars[3].substr(vars[3].indexOf("=")+1);
```

On prend comme acquis que l'URL est sous la forme `chrome://xulforum/content/index.xul?config=...&phpsessname=...&phpsessid=...&auteur=...`

On peut ensuite initialiser SOAP, avec le nom de la session et sa valeur :

```
initialiserSoap(auteur, phpSessName, phpSessId);
```

Enfin, lors d'un double-clic sur l'arbre, on pourra écrire, dans la fonction `ouvrirPopupSujet()` :

Récupération du contenu d'un message avant affichage par pop-up XBL

```
...
var message = obtenirMessageSoap(titre);
var mDate = message["date"].substr(8, 2);
var mMonth = message["date"].substr(5, 2);
var mYear = message["date"].substr(2, 2);
var mHours = message["date"].substr(11, 2);
var mMinutes = message["date"].substr(14, 2);
var div = divs[i];
div.id = titre;
div.setAttribute("sId", titre);
div.setAttribute("auteur", message["auteur"]);
div.setAttribute("date", mDate+"/"+mMonth+"/"+mYear+" "+
                        mHours+":"+mMinutes);
div.setAttribute("titre", message["titre"]);
var p = div.getElementsByTagName("p")[0];
p.innerHTML = message["texte"];
...
```

Les nombreux découpages de la chaîne ont pour but de transformer un champ MySQL `yyyy-mm-dd hh:ii:ss` en format plus lisible `dd/mm/yy hh:ii`. Vous remarquerez que l'on donne une valeur à l'attribut `sId`, qui sert lors de l'enregistrement d'une réponse.

La propriété `innerHTML` du paragraphe permet d'injecter directement du HTML dans le contenu du paragraphe.

ATTENTION Injections de code

Il faut être extrêmement prudent, lorsqu'on autorise l'utilisateur à rentrer des données. La fonction `strip_tags()` de PHP permet d'éviter les balises malveillantes comme `<object>`, `<script>`, ``, mais elle ne contrôle pas et c'est indiqué sur la page de manuel consultable sur le site de `php.net`, les attributs `onmouseover` et autres, qui pourraient servir à créer dynamiquement du JavaScript ajoutant du contenu malveillant.

Que faire donc ? Il faudrait créer notre propre fonction `strip_tags`. Un analyseur XML construirait une représentation du message et pour chaque balise n'autoriserait que les attributs minimaux. De plus, ceci permettrait de signaler un éventuel document XML mal formé, qui causerait une erreur lors de l'utilisation de `innerHTML` (si on lui donne la valeur « `site` », avec un guillemet fermant manquant, le message ne sera pas affiché).

Une autre alternative, beaucoup plus réalisable, serait l'utilisation d'une syntaxe wiki et de la classe `WikiRenderer` : en utilisant la syntaxe wiki et une classe qui crée dans tous les cas du XHTML conforme, on évite les erreurs XML et les injections de code malveillant.

► http://fr.php.net/strip_tags

► <http://ljouanneau.com/blog/2004/05/16/281-wikirenderer-205>

En résumé...

C'était la dernière touche à apporter au code de XUL Forum. Il est maintenant fonctionnel : vous pouvez vous amuser à poster des sujets et à vous répondre à vous-même.

Nous avons donc vu que pour implémenter une nouvelle méthode de communication, par-dessus une base de code déjà existante, il faut agir sur trois ensembles :

- l'ensemble du code serveur, avec le `SOAPServer` en PHP5 ;
- l'ensemble des routines JavaScript client : c'est le fichier `soap.js` ;
- l'ensemble du JavaScript chargé de l'interface utilisateur : `identification.js`, `forum.js`.

Pour terminer l'aventure XUL Forum, il ne reste plus qu'à emballer notre application, pour la distribuer facilement... nous allons pour cela tirer parti de la technologie `XPIInstall`.

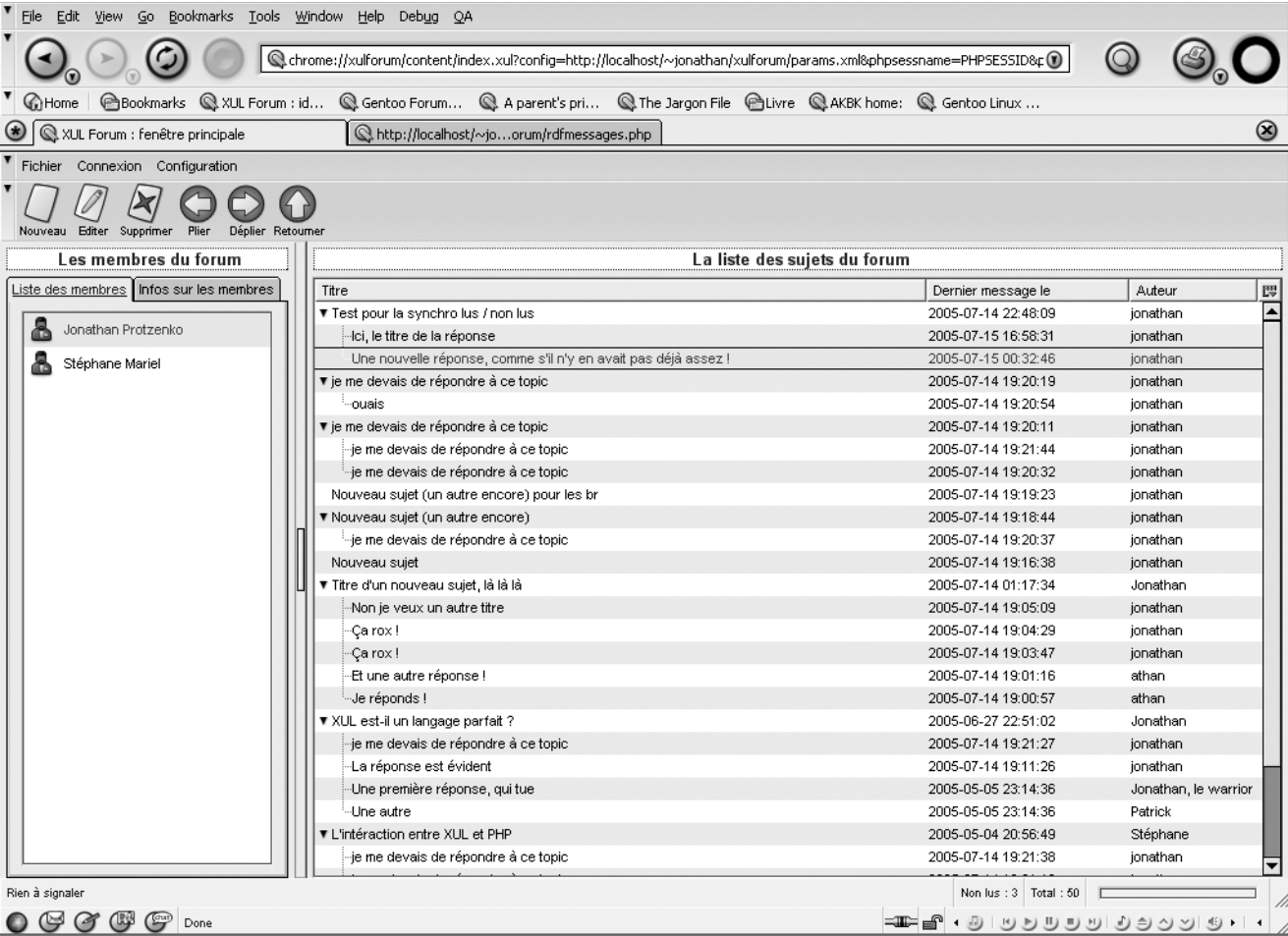


Figure 12-3 Beaucoup de sujets... pour tester le défilement !

POUR ALLER PLUS LOIN Se faciliter le travail avec WSDL

Nous avons volontairement choisi de débiter les services web via l'utilisation de l'API SOAP. Il existe cependant une autre technique : WSDL, Web Services Description Language, un autre langage XML.

WSDL sert à décrire la manière dont fonctionne un service web, comment y accéder, quels sont les paramètres à fournir, à retourner, l'URI du serveur, etc. Nous pourrions donc ainsi créer un fichier WSDL qui décrit notre xfServeur et qui servira pour les deux parties : pour PHP, nous le passerons en premier paramètre du constructeur SOAPServer. Pour Mozilla et c'est là le plus important, toutes les classes SOAP vont s'effacer au profit d'un seul objet...

Quand utiliser WSDL ?

La réponse est simple : il faut utiliser WSDL partout où c'est possible. Les traitements, surtout avec Mozilla, s'en trouvent énormément simplifiés. Si le client était en PHP, la simplification serait elle aussi énorme, avec par exemple :

```
$retour = $cClient->infosSession();
```

Les fonctions du serveur seraient enregistrées comme fonctions de l'objet client en PHP !

Cependant, le fichier WSDL n'est pas toujours présent pour un service web et en utilisant uniquement cette technologie, nous masquerions la logique SOAP qui se cache derrière : WSDL n'est finalement qu'une manière « d'envelopper » SOAP... qu'il ne faut pas pour autant perdre de vue !

Un fichier WSDL se structure généralement en quatre parties, que vous retrouverez sur le code de la page suivante.

1. La description des messages (ici nous n'avons inclus que la méthode `infosSession()`) : pour chaque message on décrit la requête et la réponse, avec à chaque fois les différents paramètres. Pour chacun des paramètres, on indique le nom et le type défini selon XML Schema : les types les plus courants sont « integer », « decimal », « boolean » ou « string ». Il est possible de créer ses propres types, comme des objets, avec des propriétés de différents types, mais ceci dépasse le cadre de XUL Forum !

2. Une section `<portType>` : pour chaque opération, on lui associe une requête (le préfixe `tns:`, comme

`targetNamespace`, indique que c'est un élément de l'espace de nommage principal) et une réponse. Il est possible d'omettre la réponse si la méthode ne retourne rien, ou même la requête si elle ne demande aucun paramètre.

3. Une section `<binding>` : on reprend les opérations décrites précédemment et on les associe à SOAP. `<soap:binding>` décrit le type d'appel et `<soap:body>` décrit la manière dont doit être formée la requête SOAP : *document style* ou *rpc style*.
4. Enfin, la section `<service>` décrit l'adresse du serveur SOAP et lui associe les `<binding>` précédents.

CULTURE Les différentes façons de présenter un message SOAP

Pour des raisons historiques, il y a plusieurs « modèles » pour structurer un message SOAP. L'article des DeveloperWorks d'IBM vous permettra d'approfondir le sujet. Car il faut bien réaliser que par l'utilisation de SOAP, on aborde tout un monde de services web, qui ouvre la porte à de nombreuses spécifications à comprendre et à maîtriser : les schémas XML, les différents types de requêtes SOAP, les types XSD...

► <http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/>

Modifications côté PHP

Avec PHP, nous pourrions écrire directement :

```
$s = new SOAPServer("./xfServeur.wsdl");
```

Il n'est plus nécessaire de préciser l'URI de ce serveur : il est directement rentré dans le fichier WSDL.

De plus, un client PHP s'utiliserait de cette manière :

```
$c = new SOAPClient("http://
    www.xulforum.org/xfServeur.wsdl");
$c->infosSession();
$c->identification("uid=...", "****");
```

Le gain de simplicité est flagrant !

Ce fichier explique à JavaScript et PHP comment fonctionne notre serveur SOAP, ce qui est à l'origine de la simplification de notre code.

```
<?xml version="1.0" ?>
<definitions name="xfDefinitions"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://www.xulforum.org/xfServeur"
  xmlns:tns="http://www.xulforum.org/xfServeur"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:monXs="http://www.xulforum.org/monXs"
>

  <message name="infosSessionRequete" />
  <message name="infosSessionReponse">
    <part name="phpsessid" type="xs:string" />
    <part name="phpsessname" type="xs:string" />
    <part name="identOk" type="xs:boolean" />
  </message>

  <portType name="xfServeurPortType">
    <operation name="infosSession">
      <input message="tns:infosSessionRequete" />
      <output message="tns:infosSessionReponse" />
    </operation>
  </portType>

  <binding type="tns:xfServeurPortType" name="xfServeurBinding">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="infosSession">
      <soap:operation soapAction="infosSession" />
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:xfServeur:infosSession"
          use="encoded" />
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:xfServeur:infosSession"
          use="encoded" />
      </output>
    </operation>
  </binding>

  <service name="xfServeurService">
    <port name="xfServeurPort" binding="tns:xfServeurBinding">
      <soap:address location="http://localhost/~jonathan/xulforum/soap.php" />
    </port>
  </service>
</definitions>
```

Un gain de simplicité incomparable avec Mozilla

Du côté de JavaScript, un fichier `wsdl.js` ressemblerait, à peu de choses près, à ceci :

```
var gProxy = null;

function initialiserWsd1(){
    var listener = {
        onLoad: function (pProxy) {
            gProxy = pProxy;
        },
        onError: function (pError){
            dump("Erreur : " + pError);
        },
        infosSession : function (phpsessname,
            phpsessid, identOk) {
            //traitement...
        }
    };
};

try {
    netscape.security.PrivilegeManager.
        enablePrivilege("UniversalBrowserRead");
    var factory = new WebServiceProxyFactory();
    factory.createProxyAsync("http://localhost/
        ~jonathan/xulforum/xfServeur.wsdl",
        "xfServeurPort", null, true, listener);
} catch (e) {
    dump("Impossible de créer le proxy Wsd1")
}

function test() {
    initialiserWsd1();
    if (!gProxy) {
        dump("L'initialisation n'est pas terminée,
            veuillez essayer ultérieurement");
        return;
    }
    dump(gProxy.infosSession());
}
test();
```

On pourrait alors transformer toute la partie SOAP en partie WSDL. Il faudrait découper chaque fonction synchrone en deux : d'une part, toutes les actions à effectuer avant l'appel de la méthode distante, bien évidemment, d'autre part, les actions conséquentes au retour WSDL.

On voit bien ici que les opérations ne sont pas du tout synchrones : la méthode du listener `infosSession` n'est appelée que lorsque les retours de la méthode distante arrivent, ce qui peut être plusieurs secondes après l'appel original. Un traitement asynchrone permet une meilleure expérience utilisateur... mais plus de complexité dans les scripts.

Un premier problème qui se pose serait de savoir, une fois que l'on a cliqué sur « OK » à l'identification, comment déterminer si LDAP et SOAP ont tous deux réussi leur identification. Il faudrait alors repenser différemment les fonctions. Deux variables globales, `gLdapOk` et `gSoapOk` seraient mises toutes les deux à `false` juste après la pression sur « OK ». Ensuite, on lance les deux identifications. LDAP, une fois son identification réussie, met `gLdapOk` à `true` et appelle `passerPageSuivante()`. SOAP procède de même mais c'est `gSoapOk` qui est mis à `true`. Ensuite, `passerPageSuivante` commencera par :

```
if (!gLdapOk && gSoapOk){
    return;
```

Les autres fonctions ne devraient quant à elles pas poser de problèmes majeurs.

Les limitations de Mozilla : bug 271560

Il n'est cependant pas possible de tout récrire en « version WSDL ». D'abord, parce que ce serait fort long et ne ferait qu'alourdir un chapitre déjà conséquent et surtout, parce que Mozilla souffre d'un bogue qui fait planter le navigateur à chaque tentative d'initialisation d'un `WebServiceProxyFactory`.

Au moment de finaliser ce chapitre, le bogue n'est toujours pas résolu. Un correctif est disponible sur la page du bogue sur le site BugZilla et semble résoudre le problème, mais il n'a pas été appliqué sur les sources. Concrètement, tant que ce patch ne sera pas intégré dans le CVS, le bogue perdurera dans Firefox, Thunderbird, Mozilla... Il a cependant été marqué comme bloquant la sortie de la version 1.8 bêta 4, qui est planifiée pour le courant de l'été. Pour finir la partie « programmation » du livre, ce sont un peu les limites de Mozilla que nous atteignons...

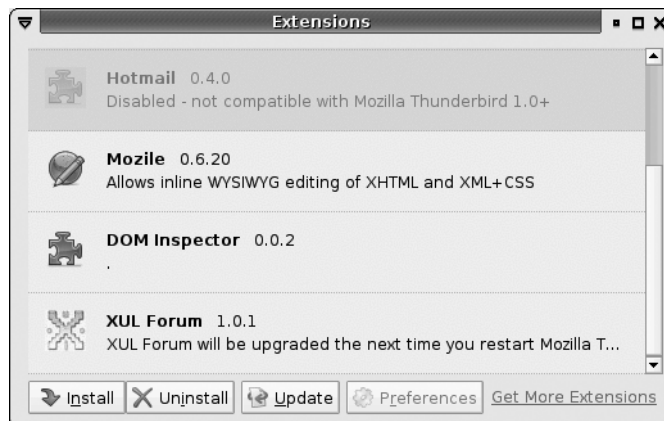
Alors, que penser ? Faiblesse, ou force de Mozilla ? De toute évidence, plutôt une force... car ce sont des utilisateurs « normaux », qui écrivent du JavaScript, qui sont capables de rapporter ces bogues. Et la communauté se penche sur le bogue (voir d'ailleurs la discussion qui a suivi ce rapport) et propose des solutions. Alors bien sûr, la réactivité n'est pas toujours idéale. Mais il y a d'autres préoccupations, plus critiques, pour assurer le bon déroulement de la route vers Firefox 1.5...

Vous aussi, si vous pensez avoir heurté un bogue de Mozilla lors de votre développement, vous pouvez en faire part aux développeurs par l'intermédiaire de BugZilla. Il y a cependant quelques règles de politesse à respecter : vérifiez très soigneusement que votre bogue n'a pas déjà été reporté, évitez de mettre une priorité maximale parce qu'il est sans doute embêtant mais peut-être seulement pour vous et surtout, testez le bogue sur la dernière version, datée du jour même, une « nightly build ».

► http://bugzilla.mozilla.org/show_bug.cgi?id=271560

13

chapitre



Distribution de XUL Forum avec XPInstall

Ceci sera le dernier chapitre : maintenant que tout est fonctionnel, que nous avons franchi les étapes les plus difficiles, que XPCOM a été maîtrisé... il est temps de revenir au chapitre 4, pour écrire, dans la lignée des `contents.rdf`, un fichier `install.rdf` ! Alors, nous pourrons utiliser XPInstall, « Cross Platform Install », pour installer XUL Forum sur n'importe quelle machine équipée d'un navigateur compatible.

SOMMAIRE

- ▶ Le fichier `.xpi` : composition
- ▶ Signaler des mises à jour à l'utilisateur

MOTS-CLÉS

- ▶ `install.rdf`
- ▶ fichiers `.jar`

Outils Zipper

Sous Windows, vous pourrez utiliser 7-zip et sous Unix, la commande :

```
zip -r monfichier.zip dossier1
➡ dossier2
```

vous sera fort utile.

► <http://www.7-zip.org/>

Bien sûr, l'installation ne se limite pas à un simple fichier mais dans ce chapitre, il n'y aura quasiment plus de programmation, simplement de la mise en forme : création de fichiers .zip, écriture de fichiers .rdf et... des tests !

Le fichier xulforum.xpi

Création du fichier xulforum.jar

Nous allons placer tous les fichiers de XUL Forum dans une archive, au format ZIP, mais portant l'extension .jar. En effet, la structure d'un fichier XPI qui, rappelons-le, est le format des extensions Mozilla, est la suivante :

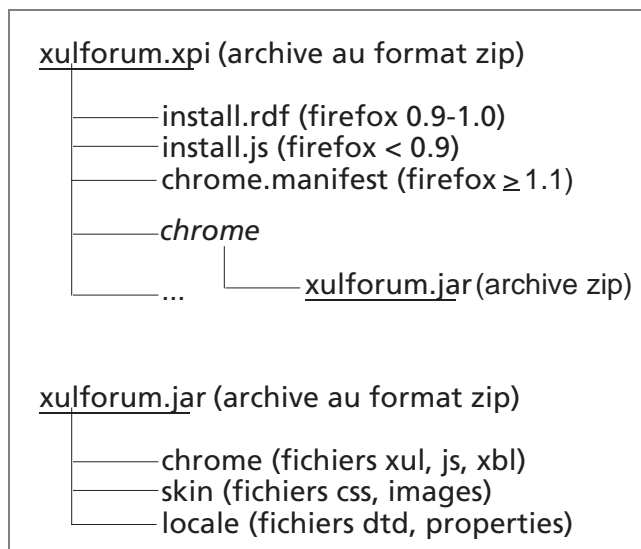
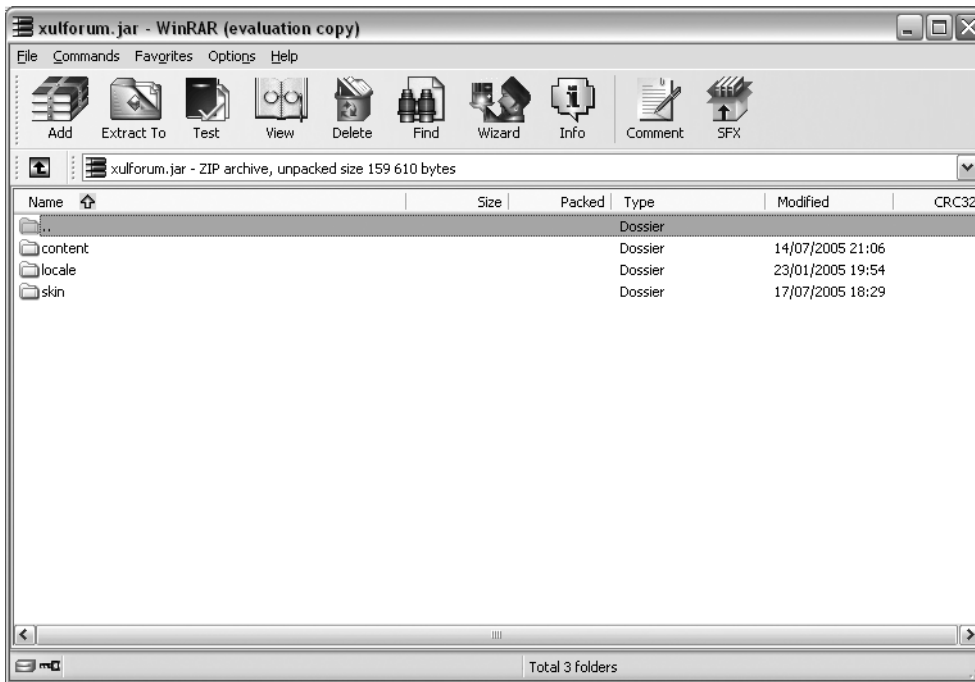


Figure 13-1 La structure de notre xpi (il y a des dossiers supplémentaires possibles, pour des composants, ou des fichiers .dll par exemple)

Une fois les dossiers content, locale et skin présents dans l'archive xulforum.jar (au besoin, créez un fichier .zip et renommez-le en .jar), il faut écrire le fichier install.rdf. Nous verrons également le fichier install.js, utilisé pour la suite Mozilla et les anciennes versions de Firefox. Pour le fichier chrome.manifest, il faut vous reporter à l'annexe expliquant comment travailler avec Firefox 1.5. Ce fichier est cependant entièrement facultatif, et même si vous utilisez Firefox 1.5, toutes les instructions données dans ce chapitre restent valides. L'annexe n'est là que pour expliquer plus en détail les nouveautés de Firefox 1.5 !

**Figure 13-2**

xulforum.jar reprend bien sûr les trois dossiers principaux

install.rdf : comment installer XUL Forum ?

Une extension est identifiée par un GUID, *Globally Unique Identifier*. On peut l'obtenir avec le programme Windows guidgen, ou sous Unix avec la commande `uuidgen`.

► <http://www.microsoft.com/downloads/details.aspx?FamilyId=94551F58-484F-4A8C-BB39-ADB270833AFC&displaylang=en>

Si vous êtes en train de créer votre propre extension, évitez de prendre le même GUID que celui donné, il sert déjà pour l'extension récupérable en ligne de XUL Forum !

```
jonathan@daisy ~ $ uuidgen
540e3545-c647-4fe0-b536-a0d728d6f641
```

Ceci nous permet d'écrire le fichier `install.rdf` (voir page suivante).

Il n'y a ici qu'un seul élément `RDF:Description` et il s'applique au fichier d'installation ❶. On doit spécifier obligatoirement :

- le GUID de l'extension ❷ : nous en avons obtenu un plus haut ;
- son nom ❸ ;

► <http://en.wikipedia.org/wiki/GUID>

Fichier install.rdf

```

<?xml version="1.0" ?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:em="http://www.mozilla.org/2004/em-rdf#">

  <Description about="urn:mozilla:install-manifest"> ❶
    <em:id>{540e3545-c647-4fe0-b536-a0d728d6f641}</em:id> ❷
    <em:name>XUL Forum</em:name> ❸
    <em:version>1.0.0</em:version> ❹

    <em:targetApplication> ❺
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.0</em:minVersion>
        <em:maxVersion>1.5</em:maxVersion>
      </Description>
    </em:targetApplication>
    <em:targetApplication> ❻
      <Description>
        <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
        <em:minVersion>1.0</em:minVersion>
        <em:maxVersion>1.5</em:maxVersion>
      </Description>
    </em:targetApplication>

    <em:description>Un forum tout en XUL !</em:description> ❼
    <em:homepageURL>http://www.xulforum.org</em:homepageURL> ❽
    <em:updateURL>http://www.xulforum.org/update.rdf
    </em:updateURL> ❾
    <em:iconURL>chrome://xulforum/skin/boutonem.jpg
    </em:iconURL> ❿

    <em:file> ⓫
      <Description about="urn:mozilla:extension:file:xulforum.jar">
        <em:package>content</em:package>
        <em:locale>locale/fr-FR</em:locale>
        <em:skin>skin</em:skin>
      </Description>
    </em:file>

  </Description>
</RDF>

```

► <http://xulfr.org/wiki/ExtensionsFirefox/PackageXPI>

- sa version ❹, de préférence sous la forme x.y.z pour permettre une compatibilité descendante avec des versions antérieures de Firefox (voir à ce propos la page du wiki de XulFR) ;
- les applications ❺ ❻ auxquelles l'extension est destinée :
 - d'abord, Firefox (identifié grâce à son GUID) : on veut une version comprise entre 1.0 et 1.5 (les deux sont incluses). Bien sûr, lorsque nous aurons à gérer la compatibilité entre les deux pour le widget XBL, il faudra faire deux paquets, dont un compatible 1.0 avec `<maxVersion>1.0</maxVersion>` et `<minVersion>1.0</minVersion>`

- ensuite, Thunderbird : l'application est toujours compatible pour les deux, grâce à l'astuce des overlays ; on peut donc indiquer de même le GUID de Thunderbird et les versions minimale et maximale.
- ensuite viennent des éléments ⑦ qui ne sont pas indispensables, comme une description qui s'affichera dans le gestionnaire d'extensions, avec à ses côtés :
 - une URL ⑧ pour la page du projet ;
 - une URL ⑨ vers un fichier RDF spécial capable d'indiquer si une nouvelle version est disponible ;
 - enfin, une URL ⑩ vers une icône, toujours à destination du gestionnaire d'extensions ;
- le dernier élément ⑪ est un fichier et il décrit (RDF:Description) le contenu du fichier .jar que nous avons créé précédemment :
 - <package> pour le contenu : le chemin est indiqué depuis la racine du paquet jar ;
 - <skin> pour le thème ;
 - <locale> pour les fichiers de langue : on aurait aussi pu rajouter les fichiers de langue anglaise.

Il y a un bon nombre d'autres paramètres possibles : des contributeurs éventuels, un auteur... il y a d'ailleurs une page dédiée à ce propos sur le site mozilla.org.

► <http://www.mozilla.org/projects/firefox/extensions/packaging/extensions.html>

En ce qui concerne les GUID, les principaux à retenir sont ceux de Firefox et de Thunderbird : ils sont indiqués dans le fichier `install.rdf`. Si par exemple vous voulez créer une extension pour un autre produit, comme l'éditeur de pages web NVU, il vous faudra récupérer son GUID (en l'occurrence, {136c295a-4a5a-41cf-bf24-5cee526720d5}).

L'utilisation d'un fichier RDF est liée à l'existence du gestionnaire d'extensions dans Firefox/Thunderbird. Ceci explique d'ailleurs le raccourci `em` (comme *extensions manager*) utilisé pour l'espace de nommage XML.

À ce stade, vous pouvez tester l'installation (Firefox et Thunderbird 1.x). La procédure est assez simple :

- vérifiez bien que `xulforum.jar`, lorsqu'il est dézippé, crée trois dossiers `content`, `locale` et `skin` dans le dossier courant ;
- créez un dossier, `dist` par exemple et créez ou placez au bon endroit le dossier `dist/chrome`, les fichiers `dist/chrome/xulforum.jar` et `dist/install.rdf` ;
- toujours dans le dossier `dist`, zippez tout ce qui s'y trouve vers `xulforum.xpi` ;

Si vous utilisez Firefox :

- placez xulforum.xpi dans votre serveur web ;
- vérifiez sa configuration, éditez au besoin le fichier .htaccess et ajoutez

```
AddType application/x-xpinstall .xpi
```

- pointez Firefox vers le fichier XPI : par exemple, `http://localhost/~jonathan/xulforum.xpi`, attendez les trois secondes nécessaires, installez, redémarrez Firefox et vous pourrez ajouter le bouton dans la barre d'outils et accéder à XUL Forum.

Si vous utilisez Thunderbird, sélectionnez *Outils, Extensions, Installer* et indiquez le fichier XPI fraîchement créé. De même qu'avec Firefox, vous devrez redémarrer et ajouter le bouton approprié dans la barre d'outils pour pouvoir utiliser XUL Forum.

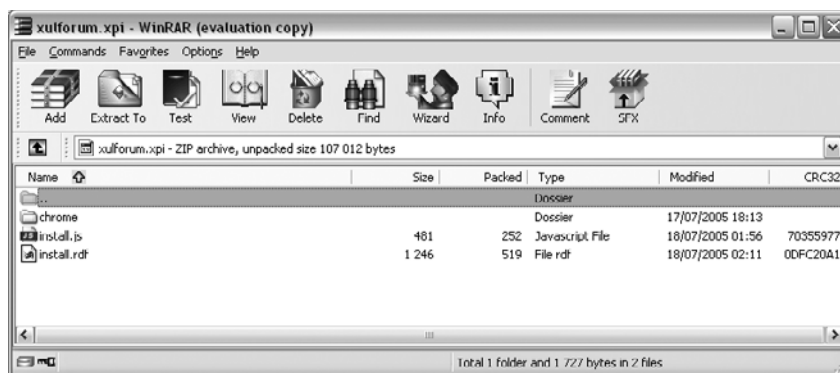


Figure 13–3
Vous devriez avoir ces
fichiers à la racine du xpi !

ATTENTION Navigateur de développement et de test

Il faut absolument séparer votre cadre de développement de votre cadre de test. Si, par exemple, vous lancez la procédure d'installation sur le même navigateur que celui qui vous sert à développer, si votre fichier `install.rdf` contient une erreur, vous aurez tout de même `chrome://xulforum/content/`, ce qui ne fera qu'ajouter à la confusion.

Le plus simple est en fait de prendre un autre navigateur et un autre profil. Par exemple, si vous développez avec Firefox, testez le XPI avec Thunderbird. Ou inversement. Vous avez aussi la solution d'utiliser deux versions séparées de Firefox : une stable pour vos tests et une instable, pour votre développement. Dans les deux cas, vous devrez créer un profil séparé, pour ne pas mêler les deux versions (les profils des versions 1.5 sont déconseillés à l'utilisation avec les versions 1.0). Vous aurez ainsi :

- une version stable, pour l'utilisation quotidienne et les tests d'installation. Firefox 1.0 par exemple. Profil : default ;
- une version de développement, disposant des dernières nouveautés et avec `chrome://xulforum/content` disponible d'office. Profil: test. Exemple: 1.5 bêta.

Pour lancer le gestionnaire de profils, il faut utiliser l'option en ligne de commande `-ProfileManager` (ou dans les menus de la suite Mozilla).

Compatibilité avec Mozilla 1.x : install.js

Pour les applications de version inférieure ou égale à Firefox 0.8 et Mozilla 1.x, qui n'utilisent pas le gestionnaire d'extensions, la procédure d'installation se fait grâce à un simple fichier JavaScript : `install.js`, à placer dans le même dossier que `install.rdf`.

Installation avec un vieux Firefox ou SeaMonkey (suite Mozilla)

```
initInstall("XUL Forum", "/jonathan/xulforum", "1.0"); ❶
var f = getFolder("Profile", "chrome"); ❷
setPackageFolder(f);
addFile("chrome/xulforum.jar"); ❸

registerChrome(PACKAGE | PROFILE_CHROME, getFolder(f,
    "xulforum.jar"), "content/"); ❹
registerChrome(LOCALE | PROFILE_CHROME, getFolder(f,
    "xulforum.jar"), "locale/fr-FR/");
registerChrome(SKIN | PROFILE_CHROME, getFolder(f,
    "xulforum.jar"), "skin/");

if (getLastError() == SUCCESS) ❺
    performInstall();
else
    cancelInstall("Une erreur est survenue.");
```

On initialise d'abord l'installation ❶, avec dans l'ordre :

- le nom de l'extension ;
- une chaîne sous la forme `/auteur/identifiant`.

On demande ensuite à obtenir le dossier `chrome` du profil de l'utilisateur ❷ : c'est la fonction `getFolder` qui retourne le chemin vers ce même dossier. On aurait pu remplacer « Profil » par « Program », mais on n'est pas toujours sûr d'avoir les droits d'accès sur le répertoire `chrome` du programme (s'il est dans `/usr/lib` par exemple) et à chaque mise à jour de Mozilla, il aurait fallu réinstaller l'extension. Il est donc plus sage d'installer l'extension dans le dossier du profil de l'utilisateur.

On indique que ceci sera le dossier du paquet et on y copie le fichier `xulforum.jar`. ❸

Les trois lignes suivantes ❹ servent à installer les dossiers `content`, `locale` et `skin`. Le premier paramètre indique le type (`PACKAGE`, `LOCALE`, `SKIN`) et spécifie de plus, grâce à un OU logique, que l'on opère dans le dossier « Profil » (ce qui est d'ailleurs le comportement par défaut de Firefox 1.x). Le second paramètre permet d'obtenir le chemin vers le fichier JAR et le dernier indique le dossier du fichier JAR qui est concerné. Enfin, si tout est bon ❺, on peut procéder à l'installation; sinon, on l'annule en renvoyant un message d'erreur.

ALTERNATIVE Se passer des fichiers .jar

Il est possible de ne pas employer de fichier JAR : on utiliserait alors `addFolder` au lieu de `addFile` dans `install.js`. Pour le fichier `install.rdf`, il faudrait faire référence au dossier `xulforum` et non pas au fichier `xulforum.jar`. On utiliserait ensuite `urn:mozilla:extension:xulforum`.

Si vous utilisez la suite Mozilla pour développer, vous pouvez maintenant tester directement l'extension. Procédez à la création du fichier .xpi comme expliqué précédemment, avec les fichiers `install.js` et `install.rdf` à l'intérieur, pour plus de compatibilité : c'est terminé ! Le plus simple est ensuite d'offrir un lien vers le fichier .xpi et la suite Mozilla vous proposera automatiquement de l'installer.

En combinant les deux types d'installation et l'astuce des overlays, avec un même fichier .jar, issu des sources décrites tout au long du livre, on a un fichier XPI qui fonctionne avec SeaMonkey 1.0, Firefox 1.5 et Thunderbird 1.5 !

ALTERNATIVE Installation via JavaScript

XULPlanet propose une autre méthode d'installation, déclenchée par un script. Elle fonctionne toujours et permet de plus d'avoir sa propre fonction de retour, appelée après l'installation. En voici un exemple, fonctionnel, à placer dans le même dossier que `xulforum.xpi`.

```
<?xml version="1.0" ?>
<html>
<head>
  <title>Installation de XUL Forum</title>
  <script type="application/x-javascript">
    function retour(nom, resultat) {
      alert("Le paquet "+nom+" a été installé ; code retour : "+
        resultat+"(0 indique le succès)");
    }

    function installer() {
      var xpi = new Object;
      xpi.xulforum = "xulforum.xpi";
      InstallTrigger.install(xpi, retour);
    }
  </script>
</head>
<body>
  <strong>Pour Mozilla 1.x</strong>
  <a href="#" onclick="javascript:installer(); return false;">
    Installer
  </a>
</body>
</html>
```

Signaler des mises à jour futures

Ce sera la touche finale ; nous avons précisé un `<em:updateURL />` dans le fichier `install.rdf` : en plaçant un fichier `.rdf` correct à l'adresse indiquée, nous serons en mesure d'indiquer au gestionnaire d'extensions de Firefox ou de Thunderbird qu'une nouvelle version est arrivée. Cette méthode n'est cependant pas à utiliser avec les extensions hébergées sur le site `update.mozilla.org`, car le site possède son propre fichier de mises à jour, utilisé par défaut par Firefox s'il n'y a pas d'`updateURL` spécifié. Tout se fait via le contenu du fichier `update.rdf`.

Le fichier signalant les mises à jour

```
<?xml version="1.0" ?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:extension:{540e3545-c647-4fe0-b536-a0d728d6f641}">
    <em:updates>
      <Seq>
        <li resource="urn:mozilla:extension:{540e3545-c647-4fe0-b536-a0d728d6f641}:1.0.0" />
        <li resource="urn:mozilla:extension:{540e3545-c647-4fe0-b536-a0d728d6f641}:1.0.1" />
      </Seq>
    </em:updates>
  </Description>

  <Description about="urn:mozilla:extension:{540e3545-c647-4fe0-b536-a0d728d6f641}:1.0.0">
    <em:version>1.0.0</em:version>
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.0</em:minVersion>
        <em:maxVersion>1.5</em:maxVersion>
        <em:updateLink>http://localhost/~jonathan/xulforum.xpi</em:updateLink>
      </Description>
    </em:targetApplication>
    <em:targetApplication>
      <Description>
        <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
        <em:minVersion>1.0</em:minVersion>
        <em:maxVersion>1.5</em:maxVersion>
        <em:updateLink>http://localhost/~jonathan/xulforum.xpi</em:updateLink>
      </Description>
    </em:targetApplication>
  </Description>

  <Description about="urn:mozilla:extension:{540e3545-c647-4fe0-b536-a0d728d6f641}:1.0.1">
    <em:version>1.0.1</em:version>
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.0</em:minVersion>
```



```

    <em:maxVersion>1.5</em:maxVersion>
    <em:updateLink>http://localhost/~jonathan/xulforum-1.0.1.xpi</em:updateLink>
  </Description>
</em:targetApplication>
<em:targetApplication>
  <Description>
    <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
    <em:minVersion>1.0</em:minVersion>
    <em:maxVersion>1.5</em:maxVersion>
    <em:updateLink>http://localhost/~jonathan/xulforum-1.0.1.xpi</em:updateLink>
  </Description>
</em:targetApplication>
</Description>
</RDF>

```

On ouvre d'abord avec une description de l'extension : c'est `urn:mozilla:extension:{GUID}`. C'est en fait une séquence des différentes versions disponibles.

Pour chaque version disponible, on indique son numéro et pour chaque application à laquelle elle se destine, on indique son GUID (ici ceux de Firefox puis Thunderbird) et le fichier XPI correspondant.

Pour assurer la compatibilité avec de très vieilles versions de Firefox, on aurait pu ajouter au sein du premier `Description` :

```

<em:version>1.0.1</em:version>
<em:updateLink>
  http://localhost/~jonathan/xulforum/xulforum-1.0.1.xpi
</em:updateLink>

```

Firefox et Thunderbird sont maintenant en mesure de se mettre à jour seuls dès qu'une nouvelle version est disponible (après confirmation de l'utilisateur bien sûr).

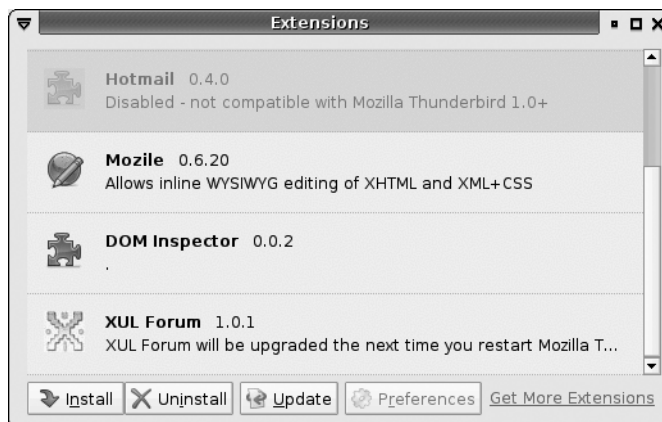


Figure 13-4
XUL Forum est dans le
gestionnaire d'extensions !

En résumé...

Il n'est pas toujours facile de maintenir une extension pour une large étendue de versions et de logiciels différents. Cependant, les nouvelles versions 1.5, même si elles introduisent une nouvelle organisation du dossier chrome (pour se passer des longs et douloureux fichiers contents.rdf), restent compatibles avec le système d'installation décrit ici. En appliquant les techniques de compatibilités évoquées, il est possible d'avoir une extension qui, au moins, s'installera sur toutes les versions de Firefox, de Thunderbird ou de Mozilla... et même du tout dernier SeaMonkey !

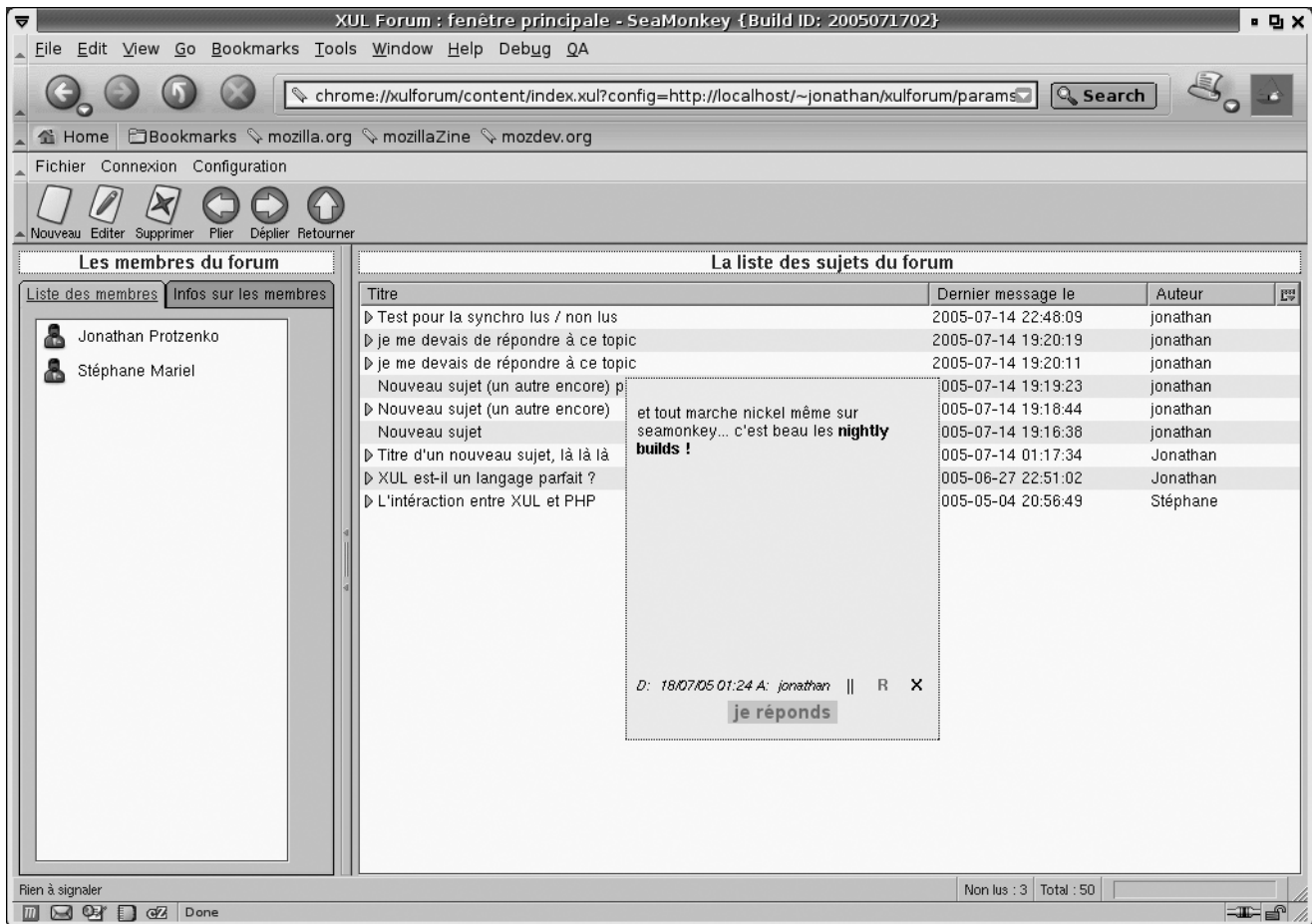


Figure 13-5 XUL Forum, sur l'une des toutes dernières versions de SeaMonkey (remarquez le logo en haut à droite qui a changé !)

Le mot de la fin

Que retenir finalement de ce long développement ? Peut-être une nouvelle conception de ce que l'on appelle les clients riches, qui, de toute évidence, sont des applications à mi-chemin entre le client lourd et le client léger et qui combinent les avantages des deux technologies.

- Le futur proche verra l'avènement des standards ouverts, passage incontournable, garants des évolutions futures : XUL et plus largement tout le XPFE sont l'incarnation de ces standards. CSS, ECMAScript, mais, surtout, XML, sont les fondations nécessaires pour assurer la solidité des prochaines applications riches.
- Le renouveau des navigateurs, rendu sensible avec l'alternative Firefox qui gagne du terrain, est aussi le signe qui montre qu'une nouvelle page de l'histoire de l'Internet est en train de s'écrire. Le monopole d'Internet Explorer vacille et peut-être XUL symbolise-t-il cette « nouvelle voie » : des applications plus riches, qui ouvrent la porte à une meilleure expérience utilisateur. Qu'est-ce que Firefox, finalement, sinon le fruit du XPFE ?
- Le XPFE a une puissance insoupçonnée au premier abord. On croit souvent (et l'on est dans le vrai si l'on se cantonne à l'antique HTML), que JavaScript et CSS ne sont que des technologies « amusantes », seulement bonnes à quelques animations appelées DHTML. Mais en plongeant dans le XPFE, en dépassant le stade de l'application test « Hello World », on peut alors réaliser toute la puissance de Mozilla.

Au cours de la conception de XUL Forum, nous avons, tout de même, réussi à implémenter les bases d'un environnement graphique (presque un « window manager »), à créer des éléments au comportement autonome, nous nous sommes connectés à divers types de systèmes (PHP, LDAP) et ce par les moyens les plus variés (XML, SOAP, API LDAP, sources RDF), notre application s'installe maintenant d'un simple clic de souris sans avoir à se soucier de problème de plate-forme... et tant de choses encore n'ont pas été abordées !

- L'apprentissage pour réaliser tout cela a pourtant été minimal, comparé à ce qui aurait été requis en utilisant un langage comme le C++... XUL est en fait de la réutilisation de savoirs déjà existants, sous une forme légère, modulaire, facile à comprendre et à faire évoluer.

Que faire maintenant ?

Je ne vois qu'une seule chose : trouver une idée originale et écrire votre propre extension !

POUR ALLER PLUS LOIN **Une version web**

Ce sera la dernière remarque... nous n'avons pas eu l'occasion de traiter d'une adaptation au format web. En effet, les limitations sont assez énormes pour envisager du « remote XUL » (ou XUL distant) : pas de composants XPCOM par exemple à disposition, pas plus que de fichiers DTD. La seule solution pour avoir accès à toutes les fonctionnalités d'une extension placée dans du chrome est de signer, à l'aide de certificats de sécurité, l'application (ou de baisser volontairement le niveau de sécurité de Mozilla, une pratique très dangereuse). Les deux premiers liens, sur les sites mozilla.org et xulfr.org vous permettront d'essayer cette technique. Le dernier décrit les limitations du XUL distant.

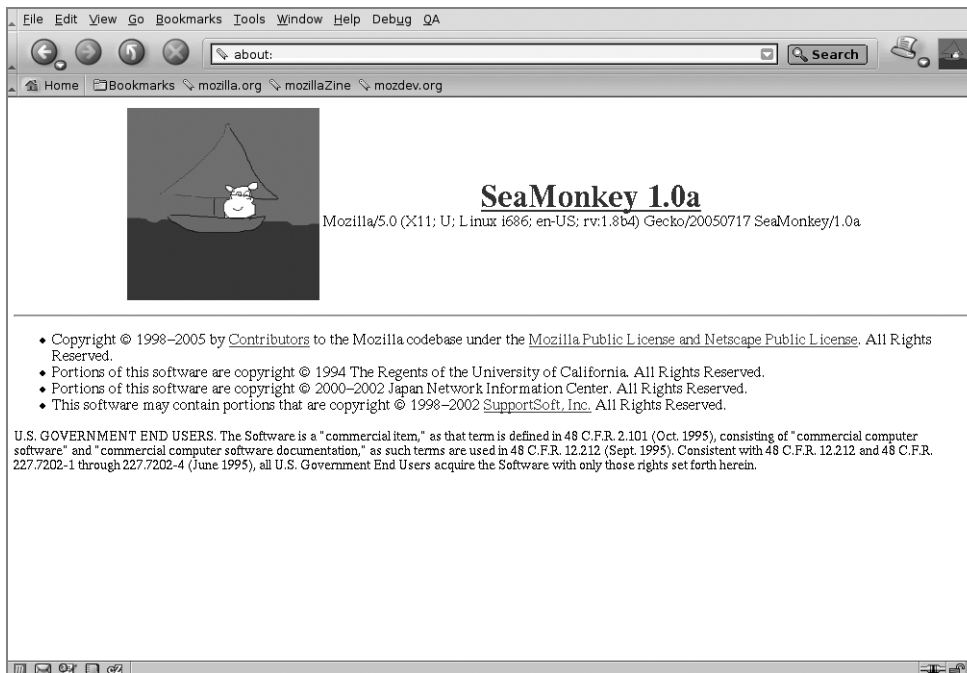
► <http://www.mozilla.org/projects/security/components/signed-scripts.html>

► <http://xulfr.org/wiki/ApplisWeb/SignerUneAppli>

► <http://xulfr.org/wiki/ApplisWeb/LimitationsDeXul>

Le futur : vers Firefox 1.5 et au-delà

A



Firefox 1.5 était initialement prévu pour le 5 juillet si l'on en croit cette roadmap : <http://wiki.mozilla.org/MozillaReleasePlanning>. Elle a ensuite été remaniée pour indiquer que Firefox 1.5 était prévu pour « Summer 2005 ». S'il n'y a pas de bogues de dernière minute, des « major regressions » ou des « blockers » comme on les appelle dans le jargon mozillien, on peut raisonnablement espérer que la sortie de cet ouvrage coïncidera avec la sortie de Firefox 1.5.

Le travail sur le chapitre 4 a été commencé alors que Firefox 1.5 n'était encore qu'une idée, ce qui explique qu'il sera principalement destiné à Firefox 1.0 : le premier paragraphe de cette annexe vise à indiquer la procédure pour transformer vos premiers fichiers XUL en extension, en supposant que vous utilisez Firefox ou Thunderbird en version 1.5. Mozilla 1.x ou SeaMonkey quant à lui utilise toujours le mécanisme des fichiers RDF.

Une autre partie de l'annexe signale les modifications mineures qu'il a fallu apporter pour passer à la version 1.5.

Il n'y a plus de contents.rdf dans Firefox 1.5 et Thunderbird 1.5

Pour développer

C'est avec un soupir de soulagement que tous ceux qui ont développé sur Firefox 1.0 liront ce titre. En effet, `chrome.rdf` n'est plus et `installed-chrome.txt` ne sert plus ! Il n'y a plus que des fichiers manifest dans le chrome.

Le fichier `xulforum.manifest`, à placer dans le dossier chrome de Firefox 1.5, ressemblera à ceci :

```
firefox/chrome/xulforum.manifest
```

```
content xulforum file:///mnt/data/Livre/xulforum/content/  
skin xulforum classic/1.0 file:///mnt/data/Livre/xulforum/skin/  
locale xulforum fr-FR file:///mnt/data/Livre/xulforum/locale/fr-FR/  
overlay chrome://global/content/customizeToolbar.xul chrome://xulforum/content/palette-overlay.xul  
overlay chrome://browser/content/browser.xul chrome://xulforum/content/firefox-overlay.xul
```

Celui pour Thunderbird sera à peu de choses près le même :

thunderbird/chrome/xulforum.manifest

```
content xulforum file:///mnt/data/Livre/xulforum/content/
skin xulforum classic/1.0 file:///mnt/data/Livre/xulforum/skin/
locale xulforum fr-FR file:///mnt/data/Livre/xulforum/locale/fr-FR/
overlay chrome://messenger/content/messenger.xul chrome://xulforum/content/messenger-overlay.xul
overlay chrome://global/content/customizeToolbar.xul chrome://xulforum/content/palette-overlay.xul
```

Ces fichiers décrivent sous une forme très simple les ajouts au chrome initial. Ils sont beaucoup plus faciles à comprendre que les fichiers RDF : d'abord, on décrit le contenu du paquet xulforum, avec le chemin vers le dossier adéquat. Ensuite, on trouve le paquet de XUL Forum, un skin - le skin classique - et le chemin vers le dossier (remarquez qu'en toute rigueur il aurait fallu placer les fichiers CSS dans le dossier skin/classic de xulforum, pour ensuite pouvoir ajouter skin/modern, etc.). Enfin, vient la locale, qui s'utilise de la même manière que l'habillage. Et en dernier, les overlays, avec d'abord le fichier qui est « overlayé » (vous trouverez le terme « overlaid » dans certains documents en anglais sur le sujet) et ensuite le fichier de XUL Forum qui contient l'overlay à appliquer.

Une fois présents dans le dossier chrome et Firefox redémarré, chrome://xulforum/content/ sera immédiatement accessible.

Tous les détails sur ce nouveau format de fichier sont disponibles sur le site mozilla.org :

► <http://www.mozilla.org/xpfe/ConfigChromeSpec.html>

Pour installer

Les fichiers install.rdf ont été eux aussi simplifiés, toujours dans l'optique d'un passage aux fichiers au format manifest. Mais attention ! Si vous suivez la « nouvelle méthode » pour l'installation de l'extension sur Firefox ou Thunderbird version 1.5, vous casserez la compatibilité avec les versions antérieures, alors qu'en gardant la méthode utilisée au chapitre 13, vous serez assuré d'être compatible avec des versions de Firefox allant de 0.9 à 1.5. En effet, Firefox 1.5 créera tout seul les fichiers au format manifest qui lui sont nécessaires, à partir du fichier install.rdf.

POUR ALLER PLUS LOIN

Le fonctionnement interne du gestionnaire d'extensions

► <http://www.mozilla.org/projects/firefox/extensions/em-changes.html>

POUR ALLER PLUS LOIN **Plusieurs paquets dans un seul fichier XPI**

Cette page du site mozilla.org vous permettra d'approfondir le sujet. Versions récentes uniquement !

► http://wiki.mozilla.org/Extension_Manager:Multiple_Item_Packages

Si vous voulez quand même installer « à la façon » 1.5, il suffit de supprimer l'élément `<em:file>` dans le fichier `install.rdf` et d'ajouter un fichier `chrome.manifest` à la racine du XPI.

xulforum.xpi/chrome.manifest

```
content xulforum jar:chrome/xulforum.jar!/content/
skin xulforum classic/1.0 jar:chrome/xulforum.jar!/skin/
locale xulforum fr-FR jar:chrome/xulforum.jar!/locale/fr-FR/
overlay chrome://global/content/customizeToolbar.xul chrome://xulforum/content/palette-overlay.xul
overlay chrome://browser/content/browser.xul chrome://xulforum/content/firefox-overlay.xul
```

Ici, au lieu de pointer vers des dossiers, on pointe vers le fichier JAR et on indique, parmi les dossiers qu'il contient, celui qui est concerné.

CULTURE Protocole JAR

Si vous êtes amené à développer une application XUL dans un contexte web, vous pourrez emballer l'ensemble de vos fichiers dans un paquet JAR, puis y accéder via l'URL `jar:http://localhost/~jonathan/xulforum/dist/chrome/xulforum.jar!/content/xulforum.xul` (URL à taper dans la barre d'adresse, avec le bon chemin vers votre fichier `xulforum.jar`).

► http://wiki.mozilla.org/XUL:Xul_Runner

Un petit nouveau... XUL Runner !

XUL Runner est un projet très prometteur : il s'agit d'un environnement autonome d'exécution pour applications XUL. En simplifié, XUL Runner est comme Firefox, mais débarrassé de tous les fichiers propres à la navigation. Il sert pour les gens qui ne veulent pas installer Firefox ou Thunderbird mais qui souhaitent tout de même pouvoir lancer des applications XUL. Il utilise des fichiers portant l'extension `.ini` (pour l'instant), extension qui pourra plus tard être renommée en `.xulapp`, afin de permettre une association entre fichiers et applications sur les systèmes Windows.

L'objectif est, à terme, de concevoir Firefox, Thunderbird, comme des applications XUL s'exécutant dans le cadre de XUL Runner. Il n'y aurait alors plus qu'un seul GRE (*Gecko Runtime Environment*) en mémoire, d'où des gains non-négligeables.

Il existe des « nightlies » de XUL Runner disponibles sur le serveur FTP de mozilla.org ; vous pouvez aussi le compiler vous-même en utilisant le script de XUL Fr. Aux dernières nouvelles, en modifiant légèrement le script de XUL Fr pour ajouter le support LDAP (avec `ac_add_options -enable-ldap`), notre application XUL Forum fonctionne intégralement dans XUL Runner !

► <http://xulfr.org/wiki/XulRunner>

Nous allons « porter » XUL Forum pour XUL Runner : c'est en fait très simple. Dans le dossier xf-xr (pour « xulforum-xulrunner »), il nous faut :

- xf-xr/application.ini
- xf-xr/chrome/chrome.manifest
- xf-xr/chrome/xulforum.jar
- xf-xr/defaults/preferences/xulforum-prefs.js

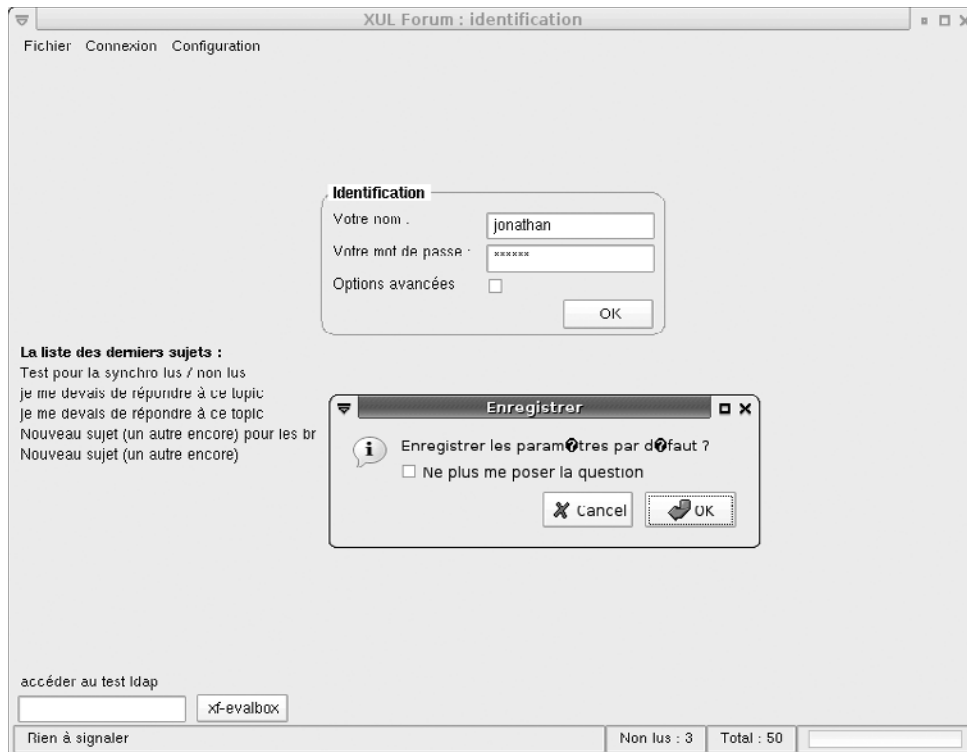


Figure A-1 XUL Forum fonctionne intégralement dans XUL Runner (sous réserve du support LDAP activé).

Vous remarquerez la différence de style des boutons « OK » et « Annuler », ainsi que les erreurs qui surviennent avec les caractères accentués.

application.ini

```
[App]
Vendor=Eyrolles
Name=XUL Forum
Version=1.0
BuildID=2005071814
ID={540e3545-c647-4fe0-b536-a0d728d6f641}

[Gecko]
MinVersion=1.8
```

► http://wiki.mozilla.org/XUL:XUL_Application_Packaging

Les noms des paramètres parlent d'eux-mêmes. Le BuildID est sous la forme yyyymmddhh. C'est le même qui est utilisé pour identifier les « nightlies » de Mozilla. Pour le widget XBL, une version de Gecko supérieure ou égale à 1.8 est requise.

chrome.manifest

Il est exactement identique à celui utilisé pour l'installation, avec les overlays en moins. On distribue XUL Forum dans un JAR, pour plus de commodité, mais on aurait aussi pu ne pas compresser l'application. Le fichier manifest aurait alors ressemblé à celui que vous utilisez dans Firefox 1.5, montré en début d'annexe.

```
content xulforum jar:xulforum.jar!/content/  
skin xulforum classic/1.0 jar:xulforum.jar!/skin/  
locale xulforum fr-FR jar:xulforum.jar!/locale/fr-FR/
```

xulforum-prefs.js

```
pref("javascript.options.showInConsole", true);  
pref("javascript.options.strict", true);  
pref("browser.dom.window.dump.enabled", true);  
pref("toolkit.defaultChromeURI",  
      "chrome://xulforum/content/xulforum.xul");
```

Les trois premières lignes ne sont que des informations de débogage. En revanche, la dernière est essentielle. En effet, il n'y a ni page de démarrage, ni barre pour saisir une adresse dans XUL Runner. Il faut donc spécifier d'une autre manière l'URL de la page par défaut de XUL Forum : c'est le rôle de la quatrième ligne.

Lancer l'application

Sous Unix, vous devrez simplement taper `/chemin/vers/le/binaire/xulrunner /chemin/vers/xf-xr/application.ini` et si vous avez bien suivi la structure précédente, l'application devrait se lancer sans aucun problème. Sous Windows, la procédure est la même, à ceci près que vous devrez utiliser le shell de Windows (accessible via le menu *Démarrer>Exécuter*, et en saisissant `cmd`) ou renommer votre fichier `.ini` en `.xulapp` et l'associer avec le binaire de XUL Runner.

Une valeur sûre : SeaMonkey

Parallèlement à Firefox 1.5 et Thunderbird 1.5 se développe (nous l'avons déjà mentionné) SeaMonkey, le successeur de la suite Mozilla, projet communautaire profitant du soutien logistique de la Fondation.

XUL Forum fonctionnera sans problème avec les dernières « nightlies » de SeaMonkey (disponibles sur le serveur FTP de mozilla.org, dans le dossier `pub/mozilla.org/seamonkey/nightly/latest-trunk/`).

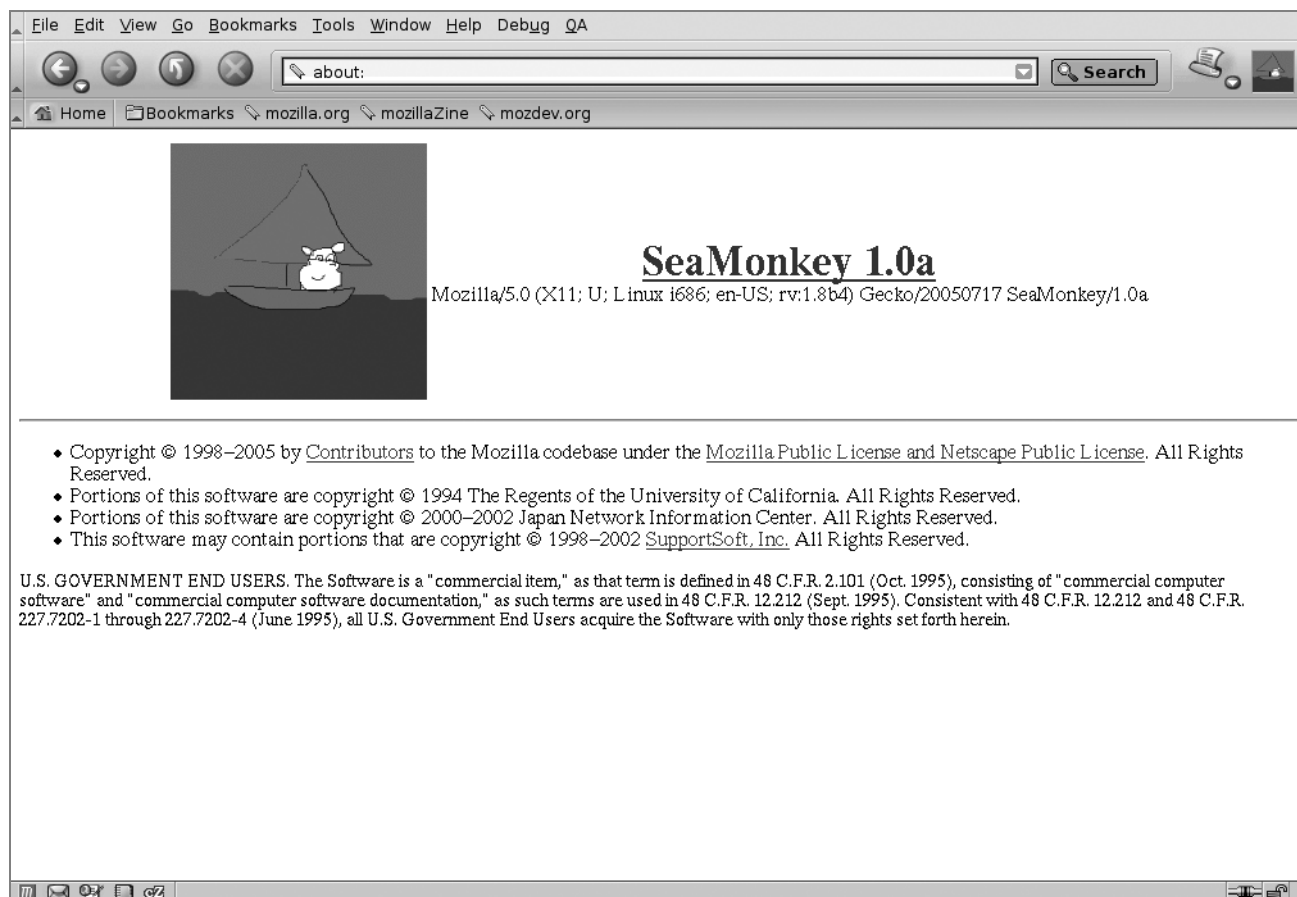


Figure A-2 Le projet SeaMonkey est encore très expérimental... tout comme son logo !

B.A.-BA Scalable Vector Graphics

SVG est un format d'image, au même titre que JPEG, PNG... La seule différence avec les deux précédents est que c'est un format d'image vectoriel ; on ne décrit pas l'image pixel par pixel, mais on en décrit les formes : une courbe, de tel angle, qui va des coordonnées (x1, y1) aux coordonnées (x2, y2), un rectangle, une ligne, un dégradé...

On n'a donc aucun effet de crénelage lorsqu'on zoome sur l'image et, surtout, SVG est au format XML, ce qui rend sa manipulation possible par DOM.

SVG est quasiment implémenté dans MiniMo, la version de Mozilla pour PDA.

- ▶ <http://weblogs.mozillazine.org/tor/archives/2005/04/index.html>

De nouvelles voies : SVG et <canvas>

Les dernières innovations ont aussi vu arriver un renouveau des API de dessin dans Mozilla : un meilleur support du SVG est désormais disponible, grâce à Cairo, et un nouveau tag HTML permet de faire du dessin en JavaScript. L'URL suivante vous permettra de tester <canvas>, le nouveau tag HTML :

- ▶ <http://developer-test.mozilla.org/presentations/xtech2005/svg-canvas/CanvasDemo.html>

POUR ALLER PLUS LOIN **Cairo en toolkit**

Dans Mozilla, les menus, les widgets, les boîtes de dialogue sont conçus en utilisant la bibliothèque graphique GTK2 (<http://www.gtk.org>), GTK signifiant « The Gimp ToolKit ». Il existe d'autres toolkits, comme GTK1 (la suite Mozilla est disponible dans les deux versions) mais surtout Cairo. Cairo est une bibliothèque de composants graphiques capable d'afficher sur de nombreux supports : PNG, X11... On peut d'ores et déjà utiliser Cairo comme toolkit expérimental (à compiler soi-même !). Le développeur principal reconnaît son affreuse lenteur, mais une coopération est prévue entre les équipes de Cairo et de Mozilla pour améliorer sa rapidité afin de, à terme, remplacer GTK2 par Cairo (le projet GTK2 prévoit de toutes façons d'utiliser aussi un jour Cairo comme moteur de rendu).

La question qui se pose est de connaître l'intérêt d'un tel changement... pour le savoir, il faut aller voir les explications du responsable de ces opérations, Robert O'Callahan :

- ▶ http://weblogs.mozillazine.org/roc/archives/2005/06/graphics_thought.html#comments
- ▶ <http://www.cairographics.org>
- ▶ <http://developer-test.mozilla.org/presentations/xtech2005/svg-canvas/>
une présentation de Xtech 2005 du même auteur, qui fait le point sur la situation, avec de très jolis exemples... à voir absolument !

Pour les scripts :

- ▶ <http://wiki.mozilla.org/SVG:Script>

Pour des exemples de fichiers SVG :

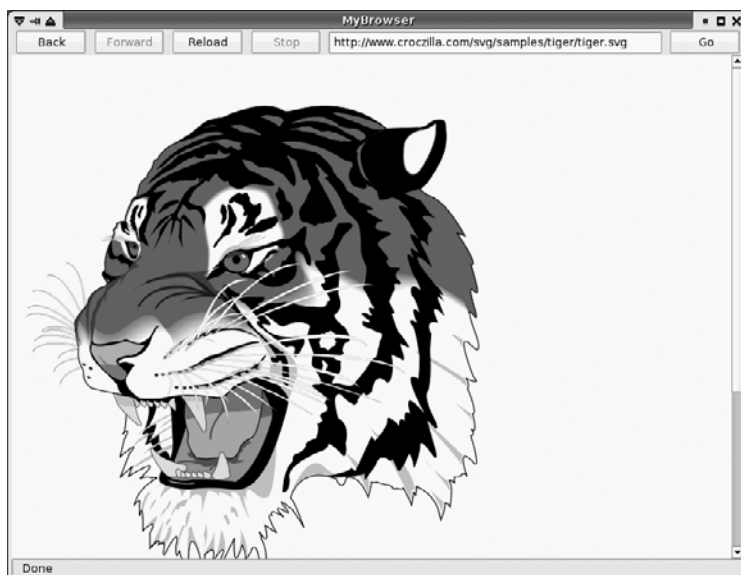
- ▶ <http://www.croczilla.com/svg/samples>

Qu'imager pour XUL Forum ?

- Grâce à SVG, un logo pour la page d'accueil, qui serait éventuellement animé grâce à JavaScript.
- Grâce à l'élément <canvas>, un écran d'identification entièrement dessiné :

- ▶ http://developer-test.mozilla.org/en/docs/Drawing_Graphics_with_Canvas

À quand une interface utilisateur de Mozilla entièrement dessinée ?

**Figure A-3**

XUL Runner lançant un « mini navigateur » (MyBrowser) qui pointe vers une image SVG (XUL Runner compilé avec support SVG)

Enfin, dernier exemple, que tout lecteur se doit d'avoir vu au moins une fois, la splendide application de dessin (oui, de dessin) vectoriel, en SVG et XBL, de Doron Rosenberg (aussi en charge du bogue WSDL). Créez vos formes, manipulez-les, le tout dans une application Mozilla.

► <http://weblogs.mozillazine.org/doron/archives/008187.html>

Des modifications mineures

Si l'on ne tient pas compte du fait que le widget XBL n'est pas compatible avec Gecko 1.7 (qui a tout de même plus d'un an !), le reste de l'application passe sans problème avec Gecko 1.8, à quelques exceptions près.

- On trouve un problème concernant le support de l'élément `<progressbar>`. Le rapport de bogue, consultable sur https://bugzilla.mozilla.org/show_bug.cgi?id=266459, mentionne un correctif qui, théoriquement, ne devrait plus être nécessaire. Cependant, pour rétablir le comportement normal de cette fonction, il nous a fallu ajouter dans le fichier `xulforum.css` ces quelques lignes (testées en dernier sur Firefox 1.5 alpha 1)

```
progressmeter[mode="undetermined"] {
    -moz-binding:
        url("chrome://global/content/bindings/progressbar.xml#progressbar-
undetermined");
}
```

-
- Étrangement, dans Thunderbird seulement, la méthode `open()` utilisée dans les overlays pour ouvrir une fenêtre pop-up contenant XUL Forum gèle l'application (la version concernée est Thunderbird 1.5 alpha 1). `openDialog()`, au comportement similaire, n'a pas ce problème, mais gèle Thunderbird en version 1.0.
 - On regrette aussi une différence des paramètres utilisés avec la méthode `getCellText()`, déjà évoquée au chapitre sur XBL.

Liste des composants XPCOM utilisés

B

Tout au long de la seconde moitié de ce livre, nous avons utilisé intensivement les composants XPCOM, ainsi que leurs interfaces. Un tableau synthétique des composants utilisés vous permettra de vous reporter au chapitre concerné et de retrouver rapidement un composant si vous aviez oublié son nom.

La procédure générale pour instancier un composant est la suivante :

- 1 Trouver le composant et aller sur le site de XULPlanet.com ; dans la référence XPCOM, repérer l'interface qui sera utile.
- 2 Instancier ce composant et demander l'interface choisie.

```
var c = Components.classes["@votre.org/composant/contractid;1"].createInstance();
var i = c.QueryInterface(Components.interfaces.nsIVotreInterface);
```

ou en raccourci :

```
var i = Components.classes["@votre.org/composant/contractid;1"].createInstance().QueryInterface(Components.interfaces.nsIVotreInterface);
```

Si c'est un service (le mot « service » se retrouvera dans le nom du composant le plus généralement), la procédure est la suivante :

```
var i = Components.classes["@votre.org/composant/cidservice;1"].getService(Components.interfaces.nsIVotreInterface );
```

- 3 Vous pouvez appeler les méthodes offertes par l'interface sur la variable i.

Tableau B-1 Les composants XPCOM utilisés dans le code de XUL Forum

Contract ID du composant	Interface(s) utile(s)	Description	Utilisé au chapitre
@mozilla.org/moz/jssubscript-loader;1	mozIJSSubscriptLoader	Pour inclure d'autres fichiers JavaScript à la manière d'un #include.	Fin du chapitre 7
@mozilla.org/rdf/rdf-service;1	nsIRDFXMLSink nsIRDFRemoteDataSource nsIRDFService	Pour contrôler la mise à jour, le rechargement d'une source RDF.	Seconde moitié du chapitre 8
@mozilla.org/preferences-service;1	nsIPrefService, nsIPrefBranch	Pour accéder aux préférences internes.	Chapitre 9
@mozilla.org/embedcomp/prompt-service;1	nsIPromptService	Boîtes de dialogue système.	Chapitre 9
@mozilla.org/network/ldap-connection;1	nsILDAPConnection	Pour se connecter à LDAP.	Chapitre 10
@mozilla.org/network/ldap-url;1	nsILDAPURL	Pour stocker les informations relatives à une connexion LDAP.	Chapitre 10
@mozilla.org/event-queue-service;1	nsIEventQueueService	Utilisation très particulière due à une implémentation en C qui nécessite une file d'attente.	Chapitre 10
@mozilla.org/xpcomproxy;1	nsIProxyObjectManager		Chapitre 10
@mozilla.org/network/ldap-operation;	nsILDAPOperation	Pour obtenir une opération LDAP. À appeler avant de procéder à un bind ou à une recherche.	Chapitre 10

Tableau B-2 Les composants XPCOM pouvant servir un jour

Contract ID du composant	Interface(s) utile(s)	Description	URL d'un guide d'utilisation
@mozilla.org/filepicker;1	nsIFilePicker	Pour choisir un fichier (boîte de dialogue système : "Ouvrir").	http://kb.mozillazine.org/Dev_:_nsIFilePicker
@mozilla.org/file/local;1	nsILocalFile	Pour accéder à des fichiers sur le disque dur.	http://kb.mozillazine.org/Dev_:_Extensions_:_Example_Code_:_File_IO
@mozilla.org/process/util;1	nsIProcess	Pour lancer des applications.	http://kb.mozillazine.org/Running_applications
@mozilla.org/widget/dragservice;1	nsIDragSession, nsIDragService	Glisser/déposer avec des éléments XUL.	http://xulplanet.com/tutorials/xultu/dragdrop.html http://xulfr.org/wiki/DragNDrop
@mozilla.org/network/socket-transport-service;1	nsISocketTransport Service	Connexions avec sockets.	http://xulplanet.com/tutorials/mozsdk/sockets.PHP
@mozilla.org/sound;1	nsISound	Sons système.	http://www.xulplanet.com/references/xpcomref/comps/c_sound1.html

Vous pourrez retrouver de nombreux exemples de code sur la base de connaissance de MozillaZine, avec d'autres composants plus techniques, non décrits ici.

► http://kb.mozillazine.org/Category:Example_code

POUR ALLER MOINS (!) LOIN **JSLib**

Les quelques composants donnés comme « pouvant être utiles » sont ceux qui possèdent au moins une page didactique expliquant leur fonctionnement, avec exemples. Vous pouvez en repérer d'autres sur la référence XPCOM de XUL Planet. Mais vous n'êtes pas certain de trouver un guide d'utilisation ! La solution la plus efficace consiste alors à rechercher un cas d'utilisation de ce composant dans un fichier .js de Mozilla, grâce à LXR.

Il existe cependant une alternative, elle s'appelle la JSLib : « JavaScript Library ». Elle a été volontairement « non-évoquée » car elle est trop enveloppante vis-à-vis de la logique interne de Mozilla : elle masque en effet tous les composants XPCOM et propose une « sur-couche ». L'exemple de code ci-dessous est tiré de la page d'introduction : il simplifie énormément le travail nécessaire si l'on devait n'utiliser que les composants XPCOM.

```
jslib.init(this);
include (jslib_file);
var file = new File("c:\\tmp\\foo.dat");
file.open("w");
file.write("This is a test\n");
file.close();
```

Une fois que vous aurez bien acquis la logique XPCOM, vous pourrez vous permettre de vous faciliter le travail avec la JSLib, qui ne vous impose pas de connaître parfaitement le fonctionnement du composant XPCOM requis. Ce n'est cependant pas une solution miracle ! C'est simplement un projet d'abstraction écrit en JavaScript et il ne couvre pas tous les composants...

► <http://jslib.mozdev.org>

C



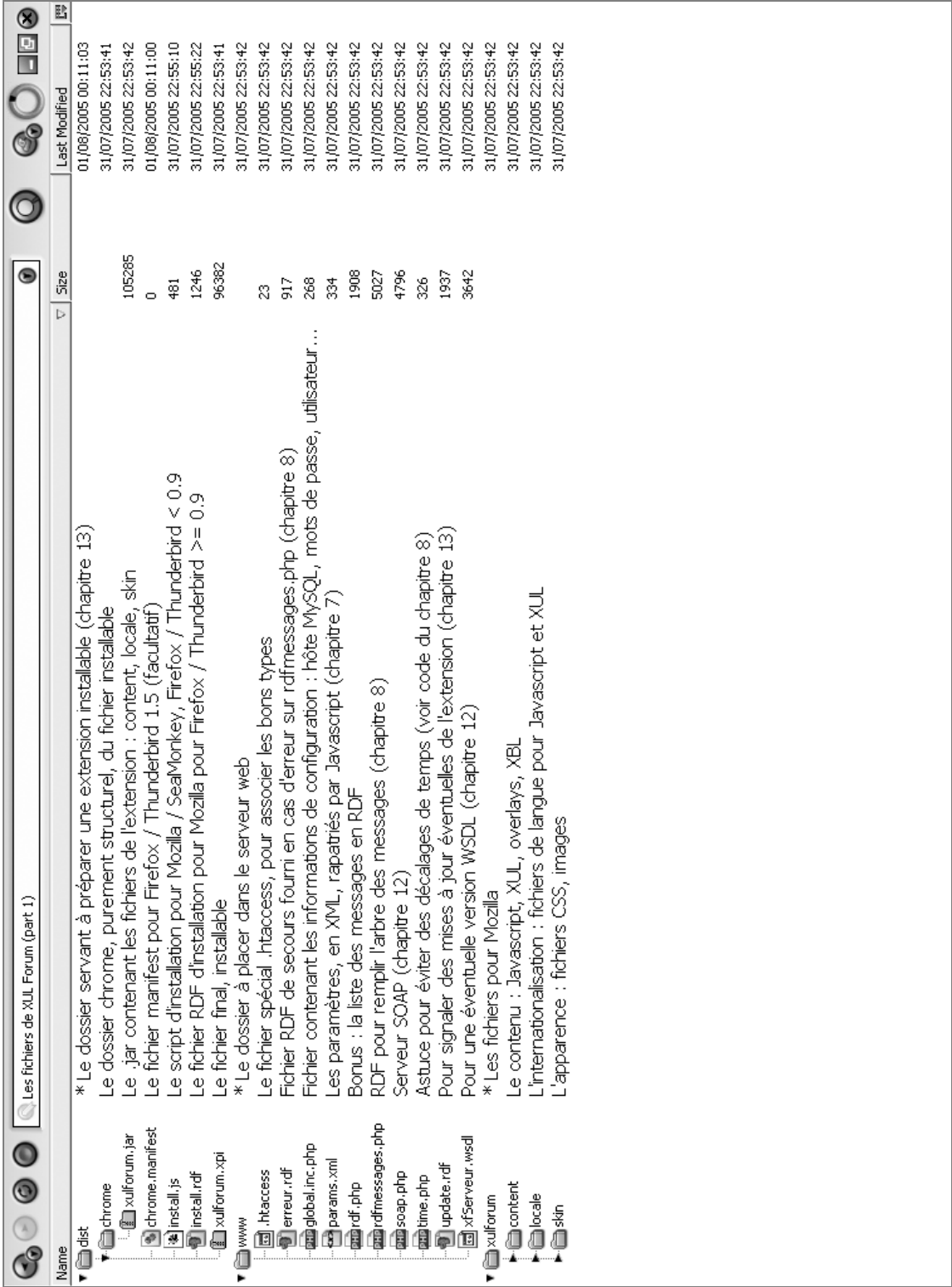


Figure C-1 Les fichiers de XUL Forum (première partie)

Les fichiers de XUL Forum (part 2)			
Name	Size	Last Modified	
xulforum		31/07/2005 22:53:42	
content		31/07/2005 22:53:42	
contents.rdf	1418	31/07/2005 22:53:42	* Le contenu
firefox-overlay.xul	679	31/07/2005 22:53:42	Le fichier RDF pour enregistrement (non utilisé dans Firefox / Thunderbird 1.5)
index.xul	1867	31/07/2005 22:53:42	L'overlay pour les boutons de la barre d'outils (Firefox uniquement)
index-barres-overlay.xul	1067	31/07/2005 22:53:42	L'écran principal du forum : liste des membres et arbre des messages
index-forum-overlay.xul	1925	31/07/2005 22:53:42	Overlay avec barres d'outils et de statut
index-membres-overlay.xul	1275	31/07/2005 22:53:42	Overlay avec l'arbre des messages
javascript		31/07/2005 22:53:42	* Les fichiers javascript
config.js	2115	31/07/2005 22:53:42	Récupérer les fichiers de configuration (chapitre 7)
forum.js	7653	31/07/2005 22:53:42	Routines de gestion de l'interface pour index.xul
global.js	742	31/07/2005 22:53:42	Routines de gestion d'erreur (chapitre 7)
identification.js	5933	31/07/2005 22:53:42	Routines de gestion de l'interface pour xulforum.xul
ldap.js	6745	31/07/2005 22:53:42	Accès au serveur LDAP (chapitre 10)
prefs.js	1652	31/07/2005 22:53:42	Accès aux préférences (chapitre 9)
rdf.js	2583	31/07/2005 22:53:42	Gestion des sources RDF (chapitre 8)
soap.js	5252	31/07/2005 22:53:42	Les routines SOAP (chapitre 12)
menus-overlay.xul	2231	31/07/2005 22:53:42	Overlay avec les menus (index.xul et identification.xul)
messenger-overlay.xul	852	31/07/2005 22:53:42	L'overlay pour les boutons de la barre d'outils (Thunderbird uniquement)
mozilla-overlay.xul	377	31/07/2005 22:53:42	L'overlay pour l'entrée Outils > Options > XUL Forum (Mozilla uniquement)
palette-overlay.xul	810	31/07/2005 22:53:42	L'overlay pour l'écran de personnalisation de la barre d'outils (Thunderbird uniquement)
xbl		31/07/2005 22:53:42	* Les fichiers XBL
bindings.xbl	10640	31/07/2005 22:53:42	Notre widget fenetreMsg
xulforum.xul	3725	31/07/2005 22:53:42	L'écran d'identification (page par défaut de chrome://xulforum/content/)
locale		31/07/2005 22:53:42	* La localisation
en-US		31/07/2005 22:53:42	* Fichiers de langue anglaise
fr-FR		31/07/2005 22:53:42	* Fichiers de langue française
contents.rdf	610	31/07/2005 22:53:42	Enregistrement par fichier RDF (pas pour Firefox / Thunderbird 1.5)
js.properties	852	31/07/2005 22:53:42	Les chaînes pour Javascript (à utiliser avec <stringbundle>)
xulforum.dtd	2443	31/07/2005 22:53:42	Les chaînes pour le fichier XUL (à utiliser avec &nomdelachaine;)
skin		31/07/2005 22:53:42	* L'apparence
bouton.png	3960	31/07/2005 22:53:42	Le bouton à trois états pour la barre d'outils
boutonem.png	2701	31/07/2005 22:53:42	Le bouton pour le gestionnaire d'extensions
boutonpetit.png	2189	31/07/2005 22:53:42	Le bouton à trois états pour la barre d'outils avec les petites icônes
contents.rdf	509	31/07/2005 22:53:42	Enregistrer le skin auprès de Firefox / Thunderbird <= 1.0
icones		31/07/2005 22:53:42	* Les différentes icônes pour la barre d'outils de XUL Forum
overlay.css	578	31/07/2005 22:53:42	Les styles utilisés dans l'overlay pour la barre d'outils
stock_person.png	1086	31/07/2005 22:53:42	La petite icône utilisée dans la liste des membres, sur index.xul
xulforum.css	3513	31/07/2005 22:53:42	Les styles globaux pour XUL Forum

Figure C-2 Les fichiers de XUL Forum (deuxième partie)

CSS : syntaxe, sélecteurs, propriétés

D

Vous l'aurez compris, CSS est crucial dans le processus de développement d'une application Mozilla. Sans CSS, il est impossible d'obtenir une boîte de dialogue « À propos » digne de ce nom, avec un titre agréable à l'œil et une mise en forme structurée, une image placée correctement... On utilise d'ailleurs souvent CSS sans même le savoir : appliquer l'attribut `orient="vertical"` sur une balise `<box>` revient tout simplement à mettre sa propriété CSS `-moz-box-orient` à `"vertical"`.

Nous nous proposons donc de dresser un aperçu très rapide de la syntaxe de base de CSS, puis nous verrons les différents sélecteurs, pour enfin effectuer un tour d'horizon rapide des propriétés les plus utilisées.

ATTENTION Exhaustivité

Nous ne détaillons ici que les éléments de syntaxe les plus courants (des unités comme `ex` ne sont par exemple pas expliquées, les compteurs non plus). Pour une référence complète, vous pourrez vous reporter au très bon ouvrage cité ci-dessous, ou aux documents du W3C, qui servent de référence à cette annexe (fondée sur la norme CSS version 2.1, voir à ce propos l'historique du chapitre 6).

► <http://www.w3.org/TR/CSS21/>

📖 CSS 2, Raphaël Goetter, Eyrolles

Syntaxe de base de CSS

La syntaxe CSS vous sera très certainement familière ; il peut néanmoins s'avérer utile de redonner les règles de base. Une déclaration CSS suit la forme suivante :

```
| sélecteur { propriété: valeur; }
```

Nous verrons les différents types de sélecteurs dans la deuxième partie de cette annexe et les différents types de propriétés dans la troisième.

Il existe différents types de valeurs :

- Les nombres, par exemple : `z-index: 3;`
- Les longueurs : elles sont composées d'un nombre suivi d'une unité ; par exemple : `border-width: 3px;`. Il existe plusieurs types d'unités :
 - Les unités relatives, c'est-à-dire exprimées en fonction d'autres unités. Les plus courantes sont `em` et `px`. `font-size: 2em;` indique que la taille de la police sera deux fois plus grande que la taille « normale », qui lui aurait été assignée en l'absence de cette déclaration. Quant à `px`, il exprime des pixels, dont la taille dépend du périphérique de visualisation (les pixels d'un écran de télévision n'ont en effet pas la même taille que ceux d'un écran d'ordinateur).
 - Les unités absolues qui ne varient jamais quelles que soient les conditions extérieures. Il y a, entre autres, `cm` (centimètres), `mm` (millimètres), `in` (*inches*, pouces anglais, 2,54 cm).
- Les pourcentages : `font-size: 200%`
- Les URL :
`list-style-image: url(http://www.example.com/puce.png);`
 (l'utilisation de guillemets est facultative)
- Les couleurs (voir la liste en figure D-1) : `color: blue;` (ce sont des mots-clés, à ne pas utiliser entre guillemets). Les autres manières de définir une couleur ont aussi déjà été vues dans ce livre : `color: #0000ff` ou `#00f` en abrégé, `rgb(0, 0, 255)` ou encore `rgb(0, 0, 100%)`.
- Les chaînes, à placer entre guillemets simples ou doubles.

Il est possible de regrouper différents sélecteurs. Ainsi :

```
| s1 { déclaration1 }
| s2 { déclaration1 }
| s3 { déclaration1 }
```

est équivalent à :

```
| s1, s2, s3 { déclaration1 }
```

4.3.6 Colors

A `<color>` is either a keyword or a numerical RGB specification.

The list of keyword color names is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow. These 17 colors have the following values:

maroon #800000	red #ff0000	orange #ffa500	yellow #ffff00	olive #808000
purple #800080	fuchsia #ff00ff	white #ffffff	lime #00ff00	green #008000
navy #000080	blue #0000ff	aqua #00ffff	teal #008080	
black #000000	silver #c0c0c0	gray #808080		

In addition to these color keywords, users may specify keywords that correspond to the colors used by certain objects in the user's environment. Please consult the section on [system colors](#) for more information.

Figure D-1

De même :

```
s1 { déclaration1 }
s1 { déclaration2 }
s1 { déclaration3 }
```

est équivalent à :

```
s1 {
  déclaration1;
  déclaration2;
  déclaration3;
}
```

Il serait impossible de tout détailler ici : les façons d'inclure une feuille de style dans un document, la priorité des styles dans la cascade, les éléments de type `block` et `inline`. Cette annexe n'est qu'un aperçu rapide ; pour vous former efficacement à CSS, il vous faudra acquérir un livre conséquent sur le sujet.

Les sélecteurs

Rappel Sélecteur !important

Veillez à ne jamais oublier ce mot-clé qui, écrit après la valeur d'une propriété, la placera le plus haut possible dans la cascade CSS, permettant ainsi d'écraser des styles spécifiés par le navigateur, le thème... Si une propriété semble ne pas s'appliquer, essayez toujours avec `!important`.

Sélecteur	Signification
*	Sélecteur universel : s'applique à tous les éléments. <code>* { font-family: Arial, sans-serif }.</code>
label	La règle s'applique à un type d'élément particulier. <code>label { font-size: 200%; }</code> double la taille de tous les éléments <code><label /></code> .
vbox label	La règle s'applique à tous les <code><label /></code> contenus dans une <code><vbox></code> . Cela inclut <code><vbox> <label /> </vbox></code> mais aussi <code><vbox><hbox><label /></hbox></vbox></code> .
vbox > label	La règle s'applique à tous les <code><label /></code> directement enfants d'une <code><vbox></code> : <code><vbox><label /></vbox></code> mais pas <code><vbox><hbox><label /></hbox></vbox></code> .
label:pseudo-classe	La règle s'applique à une pseudo-classe de l'élément <code><label></code> . Les pseudo-classes sont définies au tableau suivant.
label + description	La règle s'applique aux éléments <code><description></code> immédiatement précédés d'un <code><label /></code> (la mise en forme ne concerne pas le <code><label /></code> mais uniquement le <code><description></code>).
tree[datasources]	La règle s'applique à tous les arbres dont l'attribut <code>datasources</code> a été spécifié (quelle que soit sa valeur).
tree[datasources="rdf:null"]	La règle s'applique à tous les arbres dont la valeur de l'attribut <code>datasources</code> est exactement <code>"rdf:null"</code> .
window[persist~="screenX"]	La règle s'applique aux fenêtres pour lesquelles la position à l'écran selon l'axe des X est gardée en mémoire. S'applique à <code><window persist="screenX"></code> mais aussi à <code><window persist="screenX screenY"></code> (la liste de valeurs doit être séparée par des espaces). Dans le cas d'une liste de valeurs séparées par des tirets, on utilisera <code> =</code> à la place de <code>~=</code> .
tree#xf-index-arbre	La règle s'applique à tous les arbres dont l'identifiant est égal à <code>"xf-index-arbre"</code> . On connaît aussi <code>#xf-index-arbre</code> , forme abrégée de <code>*#xf-index-arbre</code> .
label.titre	La règle s'applique à tous les <code><label></code> dont la classe est <code>"titre"</code> . Sémantiquement équivalent à <code>label[class~="titre"]</code> . On utilise aussi <code>.titre</code> , abrégé de <code>*.titre</code> .

Les pseudo-classes sont fort utiles, mais pas toujours implémentées dans les éléments XUL. Un test rapide vous dira tout de suite si l'effet que vous recherchez est possible ou non.

Pseudo-classe	Description
<code>first-child</code> (exemple : <code>vbox:first-child</code>)	Attention ! Ceci ne désigne pas le premier élément d'une boîte verticale, mais une boîte verticale qui est le premier élément de son parent. <code>vbox:first-child</code> s'appliquera à <code><hbox><vbox /></hbox></code> mais pas à <code><hbox><label /><vbox /></hbox></code> . Pour sélectionner le premier élément fils d'une boîte verticale, on aurait utilisé : <code>vbox > *:first-child</code>
<code>link, visited</code>	Pour des éléments <code>html <a></code> . <code>:link</code> désigne un lien qui n'a pas été encore visité. À l'inverse, <code>:visited</code> désigne un lien qui a déjà été visité.
<code>hover, active, focus</code>	Pseudo-classes dynamiques : la première s'applique à un élément survolé par la souris, la seconde à un élément activé (un bouton cliqué, avec le bouton de la souris toujours pressé, par exemple), la dernière à un élément dont le focus (un <code><textbox></code> a été cliqué au moyen de la souris par exemple, et dans lequel se trouve alors le curseur).
<code>first-line, first-letter</code>	Comme leur nom l'indique, elles s'appliquent respectivement à la première ligne et à la première lettre d'un élément. <code>description:first-letter { font-weight: bold; }</code> met en gras la première lettre de chaque élément <code><description></code> .
<code>lang(nom de la langue)</code>	Pour les attributs <code>html :lang</code> ou <code>xml :lang</code> dans les documents XHTML. Permet de sélectionner des éléments en fonction de leur langue.
<code>before, after</code>	Servent à créer automatiquement du contenu. elles s'utilisent de la manière suivante : <code>p:before { content: "ce qui suit est un paragraphe"; }</code> . <code>body:after { content: "Le document HTML est fini"; }</code> . Les autres déclarations spécifiées dans <code>p:before</code> ne s'appliqueront qu'à la chaîne "ce qui suit..." tandis que celles spécifiées dans <code>p</code> s'appliqueront au paragraphe et à la chaîne "ce qui suit...".

Les propriétés utiles en CSS 2.1

Encore une fois, nous n'évoquerons ici que les propriétés pouvant servir avec XUL ; seules leurs valeurs les plus usuelles sont données ici. La plupart du temps, il est possible de leur assigner une valeur « inherit », qui est celle fournie par un sélecteur placé plus haut dans la cascade CSS.

Positionnement et tailles

Propriété	Valeurs courantes	Description
<code>top, left</code>	Longueur ou pourcentage	Définit la distance au bord supérieur de la page (<code>top</code>) et au bord gauche de la page (<code>left</code>) ; il existe aussi <code>right</code> et <code>bottom</code> .
<code>width, height</code>	Longueur ou pourcentage	Définit largeur et hauteur d'un élément.

Propriété	Valeurs courantes	Description
max-width, max-height	Longueur ou pourcentage	Définit largeur et hauteur maximales d'un élément. Utilisée dans XUL Forum pour donner une largeur maximale au titre d'un pop-up sujet (le texte en trop est remplacé par « ... »). Leur équivalent pour des minima s'obtient en changeant max-width en min-width et max-height en min-height.
position	absolute fixed static relative	Définit une position absolue, c'est-à-dire place un élément au pixel près. Pour une explication plus détaillée, consulter la série des trois articles de qualité « Initiation au positionnement CSS » d'Openweb : http://openweb.eu.org/css/ .
z-index	Entier	Définit la position de l'élément sur l'axe vertical, allant de l'écran vers les yeux du lecteur. Une valeur supérieure indique une position plus en avant, donc au-dessus des cadres dont le z-index est inférieur. Utilisée pour superposer les pop-ups sujet dans XUL Forum.

Bordures, marges, marges internes

Propriété	Valeurs courantes	Description
margin	Longueur ou pourcentage, au nombre de 1 ou 4	Si il n'y a qu'une valeur, c'est la largeur de la marge générale autour de l'élément qui est spécifiée. S'il y en a quatre, cela revient à écrire : margin = margin-top margin-right margin-bottom margin-left (soit le sens des aiguilles d'une montre).
padding	Longueur ou pourcentage, au nombre de 1 ou 4	Même syntaxe que pour la marge ci-dessus, sauf que padding représente la marge interne (voir l'illustration du chapitre 6).
border-width	thin medium thick OU longueur, au nombre de 1 ou 4	Épaisseur de la bordure : fine, médium ou épais, ou bien longueur CSS. S'il y a quatre valeurs, l'ordre est le même : top right bottom left.
border-color	Couleur CSS, 1 ou 4 valeurs	Couleur de la bordure.
border-style	none, dotted, dashed, solid, double, groove, ridge...	Style de la bordure : aucune, pointillés, tirets, trait simple, trait double, « enfoncée » dans l'arrière-plan, « dépassant » de cet arrière-plan...
border	border-width border-style border-color	Raccourci. Les quatre sous-versions border-top border-right border-bottom border-left existent aussi.

Les propriétés visuelles

Propriété	Valeurs courantes	Description
visibility	visible hidden	Permet de dessiner ou non un élément sur la page (l'espace alloué est tout de même présent).

Propriété	Valeurs courantes	Description
clipping	auto rect(a, b, c, d)	Définit une portion de l'élément à afficher : tout l'élément (valeur auto) ou un rectangle à déterminer.
overflow	hidden visible scroll auto	Que faut-il faire lorsque le contenu d'un élément dépasse de la zone qui lui est allouée ? Le cacher, le montrer (et auquel cas le texte déborde du cadre), fournir obligatoirement des barres de défilement, ou laisser le navigateur gérer ce défilement ?

Les listes

Propriété	Valeurs courantes	Description
list-style-type	disc circle square decimal decimal-leading-zero lower-roman upper-roman lower-greek lower-latin upper-latin armenian georgian lower-alpha upper-alpha none	Le type de puce qui est utilisé pour les listes HTML : lower-latin numérottera i, ii, iii, iv, v, lower-alpha numérottera a, b, c, d, e...
list-style-position	inside outside	Définit si la puce doit être dans le bloc de texte ou en dehors. Voir l'illustration sur le <i>working draft</i> du W3C. ► http://www.w3.org/TR/CSS21/generate.html#img-list-inout
list-style-image	un URI none	Permet de fournir notre propre URL vers une image servant à générer la puce. Très utile en XUL. Sert pour les éléments de menu, les boutons de la barre d'outils, la liste des utilisateurs, l'arbre...
list-style	list-style-type list-style-position list-style-image	Permet de spécifier d'un seul coup les trois propriétés ci-dessus.

Couleurs et arrière-plans

Propriété	Valeurs courantes	Description
color	Une couleur CSS	Définit la couleur d'avant-plan d'un élément : utile pour la couleur du texte par exemple.
background-color	Une couleur CSS transparent none	Définit la couleur d'arrière-plan (attention, orthographe américaine du mot « color », et non pas le « colour » à l'anglaise). Peut être utilisée pour des fenêtres transparentes en XUL. Voir : ► http://pavlov.net/blog/archives/2005/04/canvas_and_xul.html
background-image	Un URI none	Définit l'image d'arrière-plan.

Propriété	Valeurs courantes	Description
background-repeat	repeat repeat-x repeat-y no-repeat	Faut-il répéter l'image d'arrière-plan si elle n'occupe pas tout l'espace de la page ?
background-attachment	scroll fixed	L'arrière-plan doit-il défiler avec la page ? Oui ou non
background-position	pourcentage pourcentage longueur longueur	Définit la distance au bord gauche et au bord supérieur pour le positionnement de l'arrière-plan.
background	background-color background-image background-repeat background-attachment background-position	Raccourci pour spécifier en même temps les propriétés ci-dessus.

Polices

Propriété	Valeurs courantes	Description
font-family	valeur1, valeur2...	Liste de polices à appliquer ; si la première n'est pas trouvée, le navigateur tentera d'appliquer la seconde, et ainsi de suite. La police peut être soit une valeur nominale : Arial, "Arial Black" (entre guillemets si la police est en plusieurs mots), soit une valeur générique comme : serif, sans-serif, cursive, fantasy, monospace. Il est recommandé de laisser une valeur générique en dernier lieu.
font-style	italic oblique normal	Une police marquée oblique peut provenir de l'inclinaison d'une police normale. Si le navigateur ne trouve pas de police italique correspondant, il tentera d'appliquer une police oblique.
font-variant	small-caps normal	Pour afficher en « petites majuscules », comme les titres de remarques de ces cahiers : ATTENTION.
font-weight	normal bold bolder lighter 100 200 300 400 500 600 700 800 900	normal correspond à la valeur numérique 400 et bold à la valeur 700. bolder et lighter sont respectivement plus gras et plus légers que la valeur héritée.
font-size	valeur absolue valeur relative longueur pourcentage	Une valeur absolue peut être : xx-small x-small small medium large x-large xx-large. Une valeur relative est soit larger, soit smaller. Elle se réfère à la taille de la police de l'élément parent, il en est de même pour les pourcentages. Une longueur s'exprime avec les unités vues plus haut.
font	font-style font-variant font-weight font-size / line-height font-family	Les trois premiers éléments sont facultatifs. line-height l'est aussi (mais doit être précédé d'un slash : on pourra écrire p { xx-large/120% } . line-height sert à préciser la hauteur des éléments lignes placés dans un élément de type bloc.

Effets sur du texte

Propriété	Valeurs courantes	Description
text-align	left right center justify	Ces propriétés parlent d'elles-mêmes : remplace les propriétés align du html : div.
text-decoration	underline overline line-through blink	Le texte aura une ligne en dessous (il sera souligné), au-dessus, par-dessus (texte barré), ou clignotera.
letter-spacing, word-spacing	Longueur CSS normal	Détermine un espace supplémentaire entre les lettres ou les mots.
text-transform	capitalize uppercase lowercase	Change l'aspect du texte : première lettre de chaque mot en majuscule, tout en majuscules, tout en minuscules.

Les extensions Mozilla à CSS

Mozilla propose ses propres propriétés CSS, et ses propres pseudo-classes, toutes préfixées par `-moz-` ; il existe aussi des extensions pour Opéra, commençant par `-o-`. Voir les liens ci-contre donnés sur le wiki du site XULFr, pour avoir la liste de ces propriétés.

- ▶ <http://lxr.mozilla.org/seamoney/source/layout/style/nsCSSPropList.h>
- ▶ <http://lxr.mozilla.org/seamoney/source/layout/style/nsCSSAnonBoxList.h>

Pseudo-classe	Description
<code>:-moz-tree-column</code>	Permet d'accéder aux colonnes d'un arbre (pour choisir une bordure par exemple).
<code>:-moz-tree-row(propriété1, propriété2)</code> (exemple : <code>#xf-index-arbre::-moz-tree-row(nonce)</code>) (notez le signe : répété deux fois)	Permet d'accéder aux lignes d'un arbre (toute la ligne, soit un assemblage de plusieurs cellules).
<code>:-moz-tree-separator</code>	Concerne les séparateurs (permet de redimensionner les colonnes).
<code>:-moz-tree-cell</code>	Définit une cellule en particulier.
<code>:-moz-tree-indentation</code>	Permet de choisir le « décalage » par rapport à la gauche des messages enfants d'un sujet.
<code>:-moz-tree-line</code>	Définit les lignes servant à connecter les messages à leur sujet (elles descendent du petit signe « + »).
<code>:-moz-tree-cell-text</code>	Permet d'accéder au texte d'une cellule (et donc le mettre en forme).

Il en existe d'autres que vous pourrez retrouver sur :

▶ <http://www.xulplanet.com/tutorials/xultu/treestyle.html>.

ou sur :

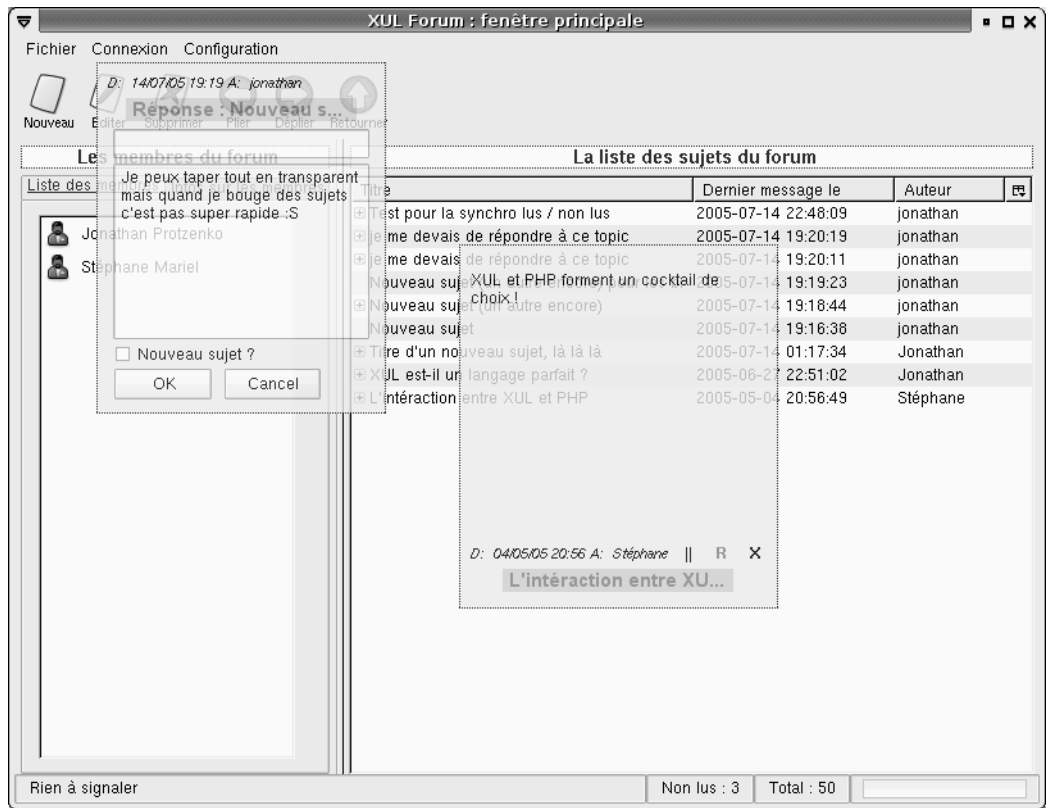
▶ http://developer.mozilla.org/en/docs/XUL_Tutorial:Styling_a_Tree

Propriété	Valeurs courantes	Description
<code>-moz-appearance</code>		Quand cette propriété prend la valeur <code>none</code> , les mises en forme qui sont appliquées sur un widget et permettent une meilleure intégration au thème général du système d'exploitation, sont alors annulées.
<code>-moz-binding</code>	URI CSS (avec une ancre)	Permet d'assigner un binding XBL à un ou des élément(s). Voir à ce propos le chapitre 11.
<code>-moz-border-radius</code> (et variantes : <code>-moz-border-radius{ bottomleft, bottomright, topleft,topright}</code>)	Longueur CSS	Pour arrondir les coins d'un cadre. Permet d'obtenir une bordure arrondie. Utilisée par défaut pour les <code><groupbox></code> qui ont les bords arrondis sur Firefox par exemple.
<code>-moz-border-{ left,right,top, bottom}-colors</code>	<code>couleur1, couleur2, couleur3...</code>	Pour créer une bordure de 3 pixels, Mozilla dessine successivement trois bordures, chacune de 1 pixel : la première sera de <code>couleur1</code> , la deuxième de <code>couleur2</code> , et ainsi de suite.
<code>-moz-box-align</code>	<code>start center end stretch</code>	Définit comment placer les éléments dans une boîte : tout au début (les éléments sont collés à gauche pour une <code><vbox></code> , en haut pour une <code><hbox></code>), les centrer, les coller à droite ou en bas, les étirer ? Par exemple : <code><hbox style="height: 500px;"><button /> <button /></hbox></code> affichage des boutons ayant une hauteur de 500 pixels. Avec <code>-moz-box-align: end</code> , les boutons gardent leur taille normale et sont placés en bas de la boîte horizontale.
<code>-moz-box-direction</code>	<code>normal reverse</code>	Pour que les éléments soient affichés dans l'ordre dans lequel ils sont placés dans le document XUL, ou pour que l'ordre soit inversé.
<code>-moz-box-flex</code>	Entier	Équivalent de l'attribut <code>flex=""</code> d'un élément XUL.
<code>-moz-box-orient</code>	<code>vertical horizontal</code>	Équivalent de l'attribut <code>orient=""</code> .
<code>-moz-image-region</code>	Rectangle CSS	Pour afficher une portion seulement d'une image. Utilisée pour les boutons de la barre d'outils.
<code>-moz-opacity</code>	Décimal	Pour rendre un élément transparent (et afficher ce qui est situé au-dessous de lui). 0 : transparent ; 1 : opaque ; entre les deux : partiellement transparent.
<code>-moz-user-focus</code>	<code>ignore normal</code>	Quand cette propriété a la valeur <code>ignore</code> , l'élément concerné ne peut pas obtenir le focus.
<code>-moz-user-input</code>	<code>disabled enabled</code>	Indique si l'utilisateur peut modifier l'élément concerné. Équivalent à <code><xxx disabled="true"></code> (où <code><xxx></code> sera souvent <code><textbox></code>).
<code>overflow</code>	<code>-moz-scrollbar-horizontal -moz-scrollbar-vertical -moz-scrollbar-none</code>	Contrôle plus finement quelles barres de défilement il faut afficher.
<code>-moz-column-count</code>	Entier	Permet une mise en forme automatique en colonnes. À appliquer sur un bloc <code><div></code> par exemple, pour forcer le texte à apparaître sous forme de colonnes. (voir http://weblogs.mozillazine.org/roc/archives/2005/03/gecko_18_for_we.html).

Propriété	Valeurs courantes	Description
-moz-column-width	Longueur CSS	Spécifie la largeur des colonnes.
-moz-column-gap	Longueur CSS	Spécifie l'espace entre des cellules.
-moz-smiley	N'est plus utilisé	Par contre, chrome://communicator/skin/smileys.css vous sera certainement utile pour utiliser les smileys inclus dans la suite Mozilla !

Figure D-2 Joli... mais lent !

La propriété `-moz-opacity` est **très** amusante, mais elle pose de gros problèmes de performance... dans le cas de XUL Forum. Sur la copie d'écran, le déplacement des sujets est très saccadé. À n'utiliser que sur des éléments fixes !



Référence des éléments XUL graphiques

E

Cette dernière annexe vous propose une véritable référence complète des éléments XUL. En effet, la technologie XUL en elle-même n'a été que très peu utilisée : nous l'avons vue en début d'ouvrage puis, lors de quelques apparitions éphémères. Ce sont surtout les autres technologies connexes qui ont pris le pas : communication avec RDF ou services web, intégration dans un navigateur ou utilisation du protocole LDAP, distribution des logiciels avec XPIInstall... On aurait pu appeler ce livre « Les Cahiers du programmeur XPFE » !

Pour pallier ce qui peut être perçu comme un manque, et fournir la liste des éléments qui seront utiles à votre application, cette annexe établit la liste des éléments graphiques XUL : non pas des éléments abstraits comme le `<rule>` des *templates*, mais plutôt des éléments concrets, visibles, dont vous aurez peut-être besoin un jour !

Des attributs génériques, communs à tous les éléments, sont généralement applicables : `height`, `width`, `orient`, `crop`, `id`, `datasources`, `oncommand`, `value`... ils ne sont pas précisés ici, sauf lorsqu'ils ont une utilité bien précise. Ils correspondent respectivement à la hauteur de l'élément, sa largeur, son orientation s'il est un conteneur (horizontale ou verticale), la manière dont on doit traiter un texte qui déborde, son identifiant, une source RDF servant à le remplir, un gestionnaire d'événement, une valeur à utiliser dans JavaScript...

Pour chaque élément, seront spécifiés ses attributs (à écrire dans le fichier XUL) et ses méthodes (à appeler depuis JavaScript). Les attributs sont aussi, pour la plupart, accessibles depuis JavaScript (ce sont alors des propriétés), mais il n'est pas toujours possible de les modifier ! En fait nous ne distinguons pas ici les attributs (décrits dans le fichier XUL) des propriétés (accessibles par JavaScript), car le parallèle entre les deux est très important : le `label` d'un bouton est aussi bien une propriété qu'un attribut.

POUR ALLER PLUS LOIN Référence XUL Planet

Bien sûr, vous l'avez déjà compris, le site de référence reste XUL Planet qui propose des définitions complètes de tout ce qui est vu ici, mais aussi les autres éléments XBL non utilisés (comme `<destructor>`), les éléments « abstraits »...

La *XUL periodic table* est aussi fort utile, avec un aperçu des différentes manipulations possibles :

- ▶ <http://www.hevanet.com/acorbin/xul/top.xul>
 - ▶ <http://www.xulplanet.com/references/>
-

CULTURE Les différents types de boutons

Il y a quatre types de boutons, en plus du type normal :

- Les boutons checkbox, à cocher ou décocher à l'aide d'un clic de souris. Mozilla gère automatiquement le changement d'état, à moins que vous ne spécifiez `autoCheck="false"`.
 - Les boutons menu, ouvrant un menu lorsqu'on clique sur le bouton (écrire : `<button><menupopup>...</menupopup></button>`).
 - Les boutons menu-button, avec une petite flèche à droite du bouton. La flèche ouvre le `menupopup`, alors que le clic sur le bouton donne un résultat différent. S'utilise de la même manière que les boutons menu.
 - Les boutons radio, à grouper avec l'attribut `group`.
-

Ceci permet de simplifier l'accès aux informations dans cette référence. Bien sûr, si vous avez un doute, le site XUL Planet s'impose ! Les attributs qui sont marqués [RO] sont *read-only*, c'est-à-dire en lecture seule : on ne peut y accéder que par JavaScript.

<arrowscrollbox>

Il s'agit de la boîte défilante qui est utilisée lorsqu'on clique sur un dossier de la *Bookmarks Toolbar* dans Firefox. De petites flèches sont affichées en haut et en bas, cliquables, permettant de faire défiler le contenu. On peut la remplir avec des éléments `<button>` par exemple.

Méthode utile

- `scrollByIndex(d)` où `d` représente le défilement, positif ou négatif selon le sens que l'on veut lui donner.

<box>

La boîte traditionnelle... voir le chapitre 3 à propos du modèle de boîte.

Attribut utile

- `orient` : pour transformer cette boîte en boîte horizontale (par défaut) ou verticale.

<button>

Ce classique bouton peut cependant être utilisé de multiples manières... les meilleurs résultats sont obtenus avec Mozilla et le thème moderne. Firefox rend en fait assez mal les boutons activés et non activés, il vaut donc mieux préférer la traditionnelle suite Mozilla, pour ensuite essayer de « porter » tout ceci vers Firefox.

Attributs utiles

- `disabled` : quand la valeur est `true`, désactive le bouton.
- `type` : pour choisir un type de bouton spécial.
- `checked` : `true` ou `false`, pour choisir d'avance l'état du bouton (dans le cas d'un bouton checkbox ou radio).
- `group` : pour des boutons radio, il faut les grouper ; c'est le rôle de cet attribut : dans un groupe de boutons radio, un seul peut être sélectionné.
- `open` : pour savoir si le menu est par défaut ouvert ou non (`true` ou `false`).
- `label` : le titre du bouton.
- `image` : URL vers une image à afficher dans le bouton, à côté du texte ; sa position dans le cadre du bouton est à contrôler avec `orient` (valeur `horizontal` : à gauche du texte, `vertical` : au-dessus du texte) et avec `dir` (si valeur `reverse`, à droite ou au-dessous).

<browser>

Cet élément est destiné à afficher une page, à la manière d'une <iframe>, mais propose des propriétés propres à un navigateur : historique, *home-page*... Utilisable uniquement dans du chrome://. C'est en fait un élément interne, au maniement complexe : si vous visualisez une page contenant un <browser> dans un onglet de Firefox, vous remarquerez que les boutons *Précédent*, *Suivant*, *Recharger* du navigateur s'appliquent désormais à votre objet <browser>. Il faut l'utiliser dans une fenêtre séparée, et créer vos propres boutons *Précédent*, *Suivant*, *Recharger*, etc. Il faut voir pour un très bon exemple d'utilisation le programme de démonstration MyBrowser, qui fonctionne dans le cadre de XUL Runner.

Il est intéressant de noter au passage que <browser> est un widget XBL dont l'implémentation est dans `chrome://global/content/bindings/browser.xml`. Ce fichier est bien sûr plus complexe que ce que nous avons utilisé pour XUL Forum. Il montre cependant quelques astuces utiles : utilisation d'un attribut `anonid` pour pallier l'unicité des `id` (en fait une astuce similaire à notre utilisation des classes CSS), héritage de bindings (grâce à la propriété `xbl:extends`), bref du XBL de haut niveau !

Attributs utiles

- `homepage` : permet de spécifier une page de démarrage pour cet objet <browser>, à utiliser plus tard avec la méthode `goHome()`. Pour y accéder via JavaScript, on utilisera la propriété `homePage` [RO].
- `src` : pour indiquer dans le document XUL quelle page afficher.
- `canGoBack` [RO], `canGoForward` [RO] : ces deux propriétés sont vraies si le navigateur peut respectivement aller en arrière ou en avant.
- `contentDocument` [RO] : l'objet Document de la page en cours dans le <browser>.
- `contentTitle` [RO] : le titre de la page.
- `contentWindow` [RO] : l'objet window de la page en cours.

Méthodes utiles

- `goBack()`, `goForward()` : pour se déplacer dans l'historique : page précédente ou suivante.
- `loadUri(adresse, referrer, charset)` : pour aller à l'adresse spécifiée, en fournissant un *referrer* (page de laquelle on vient) et un jeu de caractères.
- `reload()`, `stop()` : pour recharger et arrêter le chargement de la page en cours.

Nous ne nous attarderons pas sur cet élément très complexe qui mériterait bien plus qu'une page de référence. Si jamais vous avez besoin d'afficher une autre page web dans votre document XUL, de suivre son chargement, de gérer un historique, <browser> permettra que les composants XPCOM s'adaptent parfaitement à cet usage !

<checkbox>

La désormais célèbre case à cocher reprend la plupart des attributs du bouton.

Attributs utiles

- `checked` : indique si cette case est cochée ou non.
- `disabled` : pour désactiver la case.
- `image` : pour lui associer une image.
- `label` : pour associer un texte à cette case à cocher.

<caption>

Spécifie le titre d'un <groupbox>, utilisé dans le chapitre 3.

Attributs utiles

- `image` : pour associer une image.
- `label` : pour associer un texte.

<colorpicker>

Permet de choisir une couleur parmi les couleurs d'une grille !

Attributs utiles

- `onchange` : pour réagir aux changements de couleur.
- `color` : pour récupérer la couleur ou en choisir une par défaut (utiliser alors la forme #RRGGBB).
- `type` : si valeur `button`, l'élément s'affichera comme un bouton dont le pop-up est la grille des couleurs.

<deck>

Pour superposer des éléments mais n'en afficher qu'un à la fois. Les éléments à superposer sont les éléments enfants du <deck>.

Attributs utiles

- `selectedIndex` : pour obtenir ou changer l'index de l'élément actuellement affiché (0 représentera le premier élément fils de <deck>, 1 le second, etc.).
- `selectedPanel` : pour récupérer cet élément.

<description>

Affiche du texte. On peut lui associer une classe CSS prédéfinie, pour avoir un formatage par défaut : `header`, `indent`, `monospace`, `plain`, `small-margin`.

Attributs utiles

- `value` : pour choisir sa valeur (la mettre plutôt en tant qu'élément fils : <description>Ici le contenu...</description>).
- `disabled` : pour désactiver le texte ; généralement, il apparaît grisé.

<dialog>

Sert à créer des boîtes de dialogue en XUL pur sans utiliser de composants XPCOM comme nous l'avons fait au chapitre 8. N'hésitez pas à consulter la page du tutoriel du site XUL Planet à ce sujet ! <dialog> remplace <window>.

Attributs utiles

- `buttons` : la liste des boutons (séparés par des virgules) à afficher dans cette boîte de dialogue ; les éléments courants sont `accept`, `cancel`, `help`.
- `onDialogaccept`, `onDialogcancel`, `onDialoghelp` : pour associer un gestionnaire d'événement à chacun des boutons.
- `title` : pour choisir le titre de la boîte de dialogue.

Propriétés utiles

- `acceptDialog()`, `cancelDialog()` : pour simuler la validation ou le refus de la boîte de dialogue, soit respectivement des clics sur les boutons *OK* ou *Annuler*.

<dialogheader>

Sert à titrer, c'est-à-dire à placer un texte dans une boîte de dialogue : on écrira `<dialog><dialogheader /></dialog>`.

Attributs utiles

- `title` : le titre du <dialogheader>.
- `description` : le texte à associer à cette boîte de dialogue.

<editor>

L'éditeur inclus dans Mozilla est extrêmement difficile à utiliser : il n'inclut en effet pas la barre d'outils permettant de mettre en forme le texte, il faut donc la fournir soi-même. La meilleure solution si vous voulez un éditeur est d'utiliser une extension comme Mozile. Vous devriez cependant trouver quelques exemples d'utilisation dans le code de Mozilla de <editor>, pour la rédaction de courriers électroniques par exemple.

► <http://mozile.mozdev.org>

Attributs utiles

- `editortype` : type d'éditeur, en HTML ou en texte.
- `src` : le fichier à éditer.

Méthode utile

- `makeEditable(type,attendre)` : rendre un éditeur éditable, avec le type `html` ou `text`. On spécifie si l'on doit attendre que le document soit chargé ou non.

<grid>

Sert à créer un tableau ; vu au chapitre 3, cet élément s'utilise avec <columns>, <column>, <rows>, <row>. On écrira :

```

<grid>
  <columns>
    <column />
  </columns>
  <rows>
    <row>
      <description value="Un élément" />
    </row>
  </rows>
</grid>

```

<grippy>

La poignée est utilisée dans un séparateur, comme on l'a vu au chapitre 4. À placer comme élément fils d'un <splitter>.

<groupbox>

Sert à dessiner une bordure (généralement arrondie) autour de ses éléments fils. Utilisé au chapitre 3.

<hbox>

Nous ne la présentons plus : c'est la célèbre boîte horizontale !

<iframe>

Pour afficher une page web dans un cadre, tout simplement.

Attributs utiles

- src : pour spécifier l'URL de la page à afficher.
- contentDocument [RO], contentWindow [RO] : respectivement les objets Document et Window correspondant à la page affichée.

<image>

Sert à afficher une image.

Attributs utiles

- src : pour spécifier l'URL de l'image à afficher.
- width, height : largeur et hauteur de l'image.

<key>

Cet élément représente une touche du clavier et soit s'associe à un élément de menu (pour en faire un raccourci de menu), ou soit s'utilise seul avec un attribut `oncommand` (pour en faire un raccourci clavier). Dans tous les cas, on écrira `<keyset><key /></keyset>`. Voir à ce propos le chapitre 8 pour plus de détails.

Attributs utiles

- `key` : une touche du clavier (généralement alphanumérique).
- `keycode` : une constante commençant par `VK_` et représentant une touche spéciale (comme `Echap`).
- `modifiers` : pour faire un raccourci avec un modifieur ; les valeurs possibles sont `ctrl`, `alt`, `meta`, `control`, `accel`.

<listbox>

Une liste d'éléments, comme la liste des utilisateurs de XUL Forum, peut prendre plusieurs formes, selon le degré de complexité voulu :

- `<listbox><listitem value="xxx" /></listbox>` permet d'afficher une liste simple, dont les lignes portent seulement un texte.
- Le deuxième exemple, plus complexe, permet d'afficher des colonnes.

Exemple de liste à plusieurs colonnes

```
<listbox>
  <listcols>
    <listcol />
    <listcol />
  </listcols>
  <listitem>
    <listcell value="Colonne 1, ligne 1" />
    <listcell value="Colonne 2, ligne 1" />
  </listitem>
  ...
</listbox>
```

- Le dernier exemple sera l'occasion de donner un titre à ces colonnes.

Exemple de liste à plusieurs colonnes et avec titre aux colonnes

```
<listbox>
  <listhead>
    <listheader label="Titre colonne 1" />
    <listheader label="Titre colonne 2" />
  </listhead>
  <listcols>
    <listcol />
    <listcol />
  </listcols>
```

ASTUCE Images dans les listes

Les classes CSS `listitem-iconic` et `listcell-iconic`, une fois respectivement appliquées aux éléments `<listitem>` et `<listcell>`, vous permettront d'utiliser leur propriété `image` pour leur associer de sympathiques graphiques comme dans la liste des membres !

```
<listitem>
  <listcell label="Colonne 1, ligne 1" />
  <listcell label="Colonne 2, ligne 1" />
</listitem>
...
</listbox>
```

Cette balise propose des méthodes de manipulation des éléments de la liste, beaucoup plus nombreuses que celles évoquées ici. La liste est à l'adresse suivante : http://localhost/~jonathan/elemref/ref_listbox.html.

Attributs utiles

- `rows` : indique le nombre de lignes à afficher.
- `disabled` : désactive l'élément.
- `seltype` : `single` ne permet de sélectionner qu'une colonne à la fois ; `multiple` permet des sélections multiples (utiliser `selType` en JavaScript).

Méthodes utiles

- `appendItem(texte de l'élément, valeur associée)` : ajoute automatiquement un élément.
- `getItemAtIndex(index)` : obtenir un élément particulier de la liste.
- `selectedItem`, `selectedItems` : index de l'élément sélectionné, ou tableau des index des éléments sélectionnés.

<menubar>

Pour créer un menu on utilisera la construction suivante (voir le chapitre 5 pour plus d'informations sur les menus, dans la création de XUL Forum).

```
<menubar>
  <menu label="Fichier">
    <menupopup>
      <menuitem label="Nouveau" />
      <menuseparator />
      <menuitem label="Quitter" />
    </menupopup>
  </menu>
</menubar>
```

Attribut utile

- `grippyhidden` : cache la poignée à gauche du menu, dans la suite Mozilla.

Attributs utiles de <menu>

- `disabled` : pour le désactiver.
- `key` : id d'un élément `<key>` pour associer un raccourci clavier à ce menu.
- `open` : pour ouvrir le menu par défaut.

Attributs utiles de <menuitem>

- `disabled` : pour le désactiver.
- `key` : id d'un élément <key> pour associer un raccourci clavier à cette entrée de menu.
- `checked` : pour cocher l'entrée de menu ou la sélectionner, dans le cas de boutons cochables, ou de boutons radio.
- `type` : checkbox ou radio selon le type d'entrée de menu que l'on veut.
- `image` : pour lui associer une image ; il faut utiliser la classe CSS `menuitem-iconic` (`menuitem-non-iconic` permet à l'inverse de supprimer la marge gauche réservée à l'image).

<menulist>

Crée une liste déroulante, à la manière d'un <select> en HTML. On mettra comme élément fils un <menupopup>.

Attributs utiles

- `disabled` : pour le désactiver.
- `editable` : pour spécifier si l'on peut entrer directement la valeur au clavier.
- `image` : pour spécifier l'URL d'une image éventuelle à afficher.
- `open` : pour ouvrir le pop-up par défaut.

<menupopup>

Il s'agit d'un pop-up amené à contenir des entrées de menu. Utilisé pour les barres de menus, les boutons de type menu, les listes déroulantes...

Attributs utiles

- `onpopuphidden`, `onpopupshown` : gestionnaire d'événement à appeler lorsque respectivement le pop-up est caché ou affiché.
- `onpopuphiding`, `onpopupshowing` : gestionnaire d'événement à appeler avant de, respectivement, masquer ou de montrer le pop-up.

Méthodes utiles

- `hidePopup()` : pour masquer le pop-up.
- `moveTo(x,y)` : pour déplacer le pop-up.
- `sizeTo(width,height)` : pour redimensionner le pop-up.

<overlay>

Répartit un contenu sur plusieurs fichiers, centralise des éléments communs à plusieurs pages, etc. Voir le chapitre 5.

<page>

Remplace `<window>`, si vous comptez utiliser votre page XUL dans une `<i>frame</i>`.

<popup>

Les attributs et les propriétés sont les mêmes que pour `<menupopup>` mais l'usage est moins spécifique. Le `<popup>` se place dans un `<popupset>` et peut être associé à un clic gauche sur un élément via son attribut `popup` ou à un clic droit via son attribut `context` (ce sera alors le menu contextuel). Il faut cependant garder à l'esprit que l'action contextuelle peut être différente du clic droit, sur Mac par exemple.

Il est d'usage de remplir un `<popup>` avec des `<menuitem>`.

<progressmeter>

Il s'agit de la barre d'avancement.

Attributs utiles

- `mode` : `undetermined` pour un avancement dont la durée n'est pas connue, `determined` pour un avancement dont la durée est connue.
- `value` : le pourcentage d'avancement.

<radio>

Le bouton radio, similaire à son confrère en HTML. On le placera toujours dans un élément `<radiogroup>` : ainsi, de tous les `<radio>` contenus dans un `<radiogroup>`, un seul pourra être sélectionné.

Attributs utiles

- `disabled` : cet élément est lui aussi désactivable.
- `label` : le texte à afficher.
- `selected` : si cet élément doit être sélectionné par défaut ou non.
- `src` : l'URL d'une éventuelle image à afficher.

<scrollbar>

Vous ne vous servirez quasiment jamais de la barre de défilement pour faire défiler du contenu, l'élément en question fournissant toujours sa solution interne. Les barres de défilement s'ajouteront toutes seules à un arbre par exemple ; si l'élément s'obstine, vous pouvez tenter un `style="overflow: auto;"`.

En revanche, si vous avez besoin de sélectionner une valeur dans une plage de données, cet élément peut se révéler utile. L'attribut `orient` permettra de l'orienter verticalement ou horizontalement.

Attributs utiles

- `maxpos` : la valeur maximale prise par cette barre de défilement.
- `curpos` : la valeur actuelle.
- `increment` : la valeur à ajouter ou à supprimer lorsqu'on clique sur les flèches ; en français, on parlerait de « pas ».
- `pageincrement` : le pas lorsqu'on clique sur la zone vide entre les flèches et le rectangle représentant la position actuelle.

<spacer>

Occupe l'espace vide ; à utiliser en combinaison avec son attribut `flex`.

<splitter>

Ce séparateur est utilisé dans XUL Forum pour distinguer la liste des membres de l'arbre des messages. Un `<grippy>` peut y être placé : ce sera la poignée qui, lorsqu'elle est cliquée, peut servir à réduire totalement un pan de l'interface.

La classe CSS `tree-splitter` est utilisée pour permettre le redimensionnement des colonnes d'un arbre. Elle a été utilisée au chapitre 5.

Attributs utiles

- `collapse` : lorsqu'on doit masquer une partie de l'interface, est-ce celle située avant (valeur `before`) ou après (valeur `after`) le `<splitter>` ?
- `resizebefore`, `resizeafter` : pour déterminer les éléments à réduire lorsqu'on bouge le `<splitter>`. Est-ce l'élément le plus près du `<splitter>` (valeur `closest`), le plus éloigné (valeur `farthest`) ou doit-on redimensionner le conteneur situé avant ou après (valeur `grow`) ?
- `state` : `open` (ouvert), `collapsed` (le pan de l'interface a été masqué), `dragging` (en train d'être redimensionné).

<stack>

Similaire à un `<deck>`, mais affiche tous les éléments en même temps.

<statusbar>

La traditionnelle barre de statut.

<statusbarpanel>

Un élément de la barre de statut : un texte (propriété `label`), une image (propriété `src`) mais pas les deux.

<stringbundle>

Généralement placé dans un conteneur <stringbundleset> pour fournir les chaînes à JavaScript en fonction de la langue.

<tabbrowser>

Contient des <browser>, mais agencés avec onglets dans un <tabbox>.

<tabbox>

Permet d'afficher des panneaux à onglets : à chacun correspond un panneau. Nous en avons utilisé un pour les onglets *Liste des membres* et *Infos sur un membre*. Cet élément est employé de la manière suivante :

```
<tabbox>
  <tabs>
    <tab label="Onglet 1" />
  </tabs>
  <tabpanel>
    <description>Ici le contenu du panneau 1</description>
  </tabpanel>
</tabpanel>
</tabbox>
```

Attribut utile de <tabpanel>

- `selectedIndex` : pour sélectionner un onglet en particulier.

Attribut utile de <tabs>

- `closebutton` : booléen, sert à créer les boutons pour fermer ou ouvrir un onglet, comme dans le navigateur.

<textbox>

La classique zone de texte, utilisée dans la vie de tous les jours par le développeur XUL ;-) !

Attributs utiles

- `disabled` : encore un élément qui peut se désactiver.
- `rows`, `cols` : pour spécifier le nombre de lignes et de colonnes.
- `multiline` : à utiliser pour avoir plusieurs lignes, comme un `<html:textarea>`.
- `maxLength` : le nombre maximal de caractères que l'on peut entrer.
- `size` : le nombre maximal de caractères que l'on peut afficher.
- `type` : pour choisir un type spécial de <textbox>.
- `selectionStart`, `selectionEnd` : pour choisir ou obtenir l'index de début ou de fin des caractères sélectionnés.

CULTURE Les différents types de zone de texte

Il y en a trois :

- `autocomplete`, pour essayer de deviner la valeur que l'utilisateur veut rentrer, à la manière des champs de formulaire HTML dont Firefox se rappelle les valeurs (passage par la documentation de XUL Planet fortement conseillé).
- `password`, pour les mots de passe (affiche des étoiles au lieu des caractères).
- `timeout` : si l'on spécifie le délai `d` dans l'attribut `timeout`, l'événement `command` sera lancé après chaque nouvelle touche rentrée, et après le délai `d`.

Propriétés utiles

- `select()` : sélectionne tout le texte.
- `setSelectionRange(début, fin)` : sélectionne le texte du caractère début au caractère fin (ce sont les index des caractères qui sont fournis).
- `textLength [RO]` : longueur du texte.

<toolbar>

Le conteneur pour les boutons de la barre d'outils est utilisé dans XUL Forum pour contenir les boutons *Nouveau*, *Éditer*... Il existe d'autres fonctions, d'autres éléments, liés à la personnalisation de la barre d'outils dans Firefox, mais ils sont utilisés de manière interne par Firefox, et le développeur « classique » ne les utilise généralement pas.

Attributs utiles

- `grippyhidden` : pour masquer le *grippy* (servant à masquer une barre d'outils) dans la suite Mozilla.

<toolbarseparator>

Pour établir une séparation entre les boutons de la barre d'outils.

<toolbox>

Le conteneur vertical, amené à contenir les différentes barres : barre de menus, barre d'outils, barre avec les favoris...

<tooltip>

Cet élément est ce que l'on appelle une infobulle en français. Il est quasi équivalent à un `<popup>`, sauf que l'on y place du texte, des images, bref du contenu plus varié que les entrées de menu du `<popup>`. On l'utilise de la manière suivante :

```
<popupset>
  <tooltip id="xf-tooltip">
    <!-- ici du texte, des images, du style, etc. etc. -->
  </tooltip>
</popupset>
...
<monElement tooltip="xf-tooltip" />
<monAutreElement tooltip="Une infobulle avec juste du texte" />
```


<tree>

L'arbre, widget central de XUL Forum. Il s'utilise, rappelons-le, de la manière suivante :

```

<tree>
  <treecols>
    <treecol id="colonne1" label="Colonne 1" />
  </treecols>
  <treechildren>
    <treeitem>
      <treerow>
        <treecell label="Cellule 1" />
      </treerow>
    </treeitem>
  </treechildren>
</tree>

```

Des exemples plus évolués, avec un second niveau de profondeur, sont disponibles au chapitre 5.

Attributs utiles

- `enableColumnDrag` : pour permettre un déplacement par glisser/déposer des colonnes.
- `hidecolumnpicker` : pour masquer le petit élément permettant de montrer/cacher des colonnes.
- `seltype` : `single` pour ne sélectionner qu'une ligne à la fois ; `multiple` dans le cas contraire.
- `rows` : le nombre de lignes à afficher.
- `selectedIndex` : l'index de la ligne sélectionnée.
- `view` : l'objet implémentant `nsITreeView` qui a servi à créer l'arbre. Il en existe plusieurs, détaillés sur la page de XUL Planet : le choix de l'objet servant à créer le contenu de l'arbre est primordial. Dans le cas de XUL Forum, c'est un RDF Content Tree, utilisé pour les générations RDF et proposant des nœuds DOM. Le choix du générateur se fait en fonction des contraintes de performance et de manipulation DOM.

<treeseparator>

Une ligne de séparation dans un arbre.

<treecol>

Une colonne d'un arbre.

Attributs utiles

- `fixed` : pour empêcher le redimensionnement de cette colonne.
- `hidden` : pour masquer cette colonne.
- `ignoreincolumnpicker` : pour empêcher cette colonne d'apparaître dans la liste des colonnes à afficher ou à masquer.

- **primary** : pour la colonne principale (généralement la première) qui contiendra un second niveau de profondeur : permet d'activer l'indentation, les petits + ou – servant pour plier ou déplier la structure.
- **src** : URL d'une image qui remplace le texte de la colonne.
- **type** : pour choisir le type de cellules contenues dans cette colonne : **text** (par défaut), **checkbox** (cases à cocher), **progressmeter** (barres de progression).

<treecell>

Une cellule d'un arbre.

Attributs utiles

- **src** : URL d'une image à afficher à côté du texte.
- **mode** : pour choisir une cellule qui est une barre de progression : **none** (affiche le texte de la cellule et non le <progressmeter>), **determined** (pourcentages exacts), **undetermined** (durée inconnue).
- **value** : le pourcentage éventuel pour la barre de progression.

<vbox>

La boîte verticale.

<window>

Une fenêtre... élément de base d'un fichier XUL !

Attributs utiles

- **width**, **height** : largeur, hauteur de la fenêtre.
- **screenX**, **screenY** : les coordonnées de la fenêtre à l'écran.
- **sizemode** : **maximized** (fenêtre maximisée), **minimized** (fenêtre minimisée) ou **normal**.

<wizard>

Ceci n'est pas un attribut magique servant à créer automatiquement des documents XUL... mais un assistant, avec des pages multiples, affichées les unes à la suite des autres ! Le tutoriel de XUL Planet donne un bon cas d'exemple. Voir <http://www.xulplanet.com/tutorials/xultu/wizard.html>

```
<wizard>
  <wizardpage>
    <description>Première page de l'assistant</description>
  </wizardpage>
  <wizardpage>
    <description>Seconde page... installation de XUL Forum ?
  </description>
  </wizardpage>
</wizard>
```

Attributs utiles

- `title` : un titre général écrasé par chaque `label` des `<wizardpage>`.
- `pagestep` : index de la page actuelle (utiliser `pageIndex` depuis JavaScript).
- `firstpage`, `lastpage` : vrai pour être à la première ou à la dernière page (utiliser `firstPage` et `lastPage` depuis JavaScript).
- `onwizardback`, `onwizardnext`, `onwizardfinish`, `onwizardcancel` : gestionnaires d'événement pour respectivement les clics sur les boutons *Précédent*, *Suivant*, *Terminer* et *Annuler*.
- `onpageshow` : lorsque la page est montrée.
- `canAdvance`, `canRewind` : lorsque ces deux propriétés sont fausses, elles désactivent les boutons *Suivant* et *Précédent* (les changer depuis JavaScript).

Méthodes utiles

- `rewind()`, `cancel()`, `advance(ID de la page)` : pour simuler un appui sur les boutons.

<wizardpage>

Une page d'assistant.

Attributs utiles

- `label`, `description` : le titre, et une courte description de cette page d'assistant.
- `pageid` : pour associer manuellement un identifiant à cette page.
- `next` : pour spécifier l'identifiant de la page suivante.
- `onpageadvanced`, `onpagerewound`, `onpagehide` : ces trois gestionnaires d'événement sont appelés au moment de passer à la page suivante, précédente, ou de masquer la page ; un `return false` permettra d'annuler l'événement.
- `onpageshow` : lorsque la page est montrée.

Index

Symboles

!important 93
<![CDATA 105
<!DOCTYPE 49
<!ENTITY 49
<?xml-overlay 58
<?xml-styleSheet 82
<canvas> 262
<checkbox> 111
<children /> 195
<constructor> 198
<content> 193
<description> 29
<dialog> 165
<div> 195
<field> 198
<grid> 34
<groupbox> 34
<handler> 202
<hbox> 32
<image> 88
<implementation> 198
<listbox> 75
<menubar> 61, 154
<overlay> 58
<progressmeter> 70, 147, 263
<property> 198
<RDF:Description> 41, 129
<RDF:li> 131
<RDF:Seq> 41, 131
<spacer> 35
<splitter> 57
<statusbar> 57
<tabbox> 75
<template> 136
<textbox> 32
<toolbar> 61, 69
<toolbarpalette> 157
<toolbox> 57
<tree> 73
@import (directive CSS) 83
@mozilla.org/event-queue-service;1 179
@mozilla.org/moz/jssubscript-loader;1 121
@mozilla.org/network/ldap-connection;1 175
@mozilla.org/network/ldap-operation;1 180
@mozilla.org/network/ldap-url;1 175

@mozilla.org/preferences-service;1 160
@mozilla.org/rdf/rdf-service;1 144
@mozilla.org/xpcomproxy;1 179

A

À la recherche du temps perdu 126
accessibilité 51
AJAX (Asynchronous JavaScript And XML) 13, 215
ajouterErreur() (fonction JavaScript) 107
Amazon Browser 5
AOL 4
application.ini 259
arbre 73
 et RDF 140
 gestionnaire d'événement 208
attributs du tag <window> 49
authentification, écran d' 32

B

barre
 d'outils 91
 personnalisation 70
 de menus 61
 de statut 70
base64 183, 186
bindings.xml 192
boîte 32
bugzilla 60, 61

C

cairo 262
-chrome, option de la ligne de commande 29
chrome.manifest 260
chrome.rdf 43, 50, 84
commentaires en XUL 56
config.js 121
constructeur 104
content (dossier) 40, 41
contents.rdf 41, 47, 83, 152
contract ID 122
CSS (Cascading Style Sheets)
 application à XUL Forum 79
 avantages 81
 couleurs 85

extensions de Mozilla 91
mise en place 82
positionnement 90
pour le widget XBL 196
présentation 80
pseudo-style 87
rôle dans le XPFE 8
syntaxe 85
cursor (directive CSS) 197

D

débogage 59
display (directive CSS) 200
DOM (Document Object Model) 100
 différentes implémentations 100
 manipulation avec JavaScript 110
DTD (Document Type Definition)
 intégration à XUL Forum 45
 rôle dans le XPFE 9
 syntaxe 46
DXULi 24

E

e4X 119
ECMAScript 100
 rôle dans le XPFE 8
entités XML 9
erreur
 affichage sur la console 59
 gestion en JavaScript 106
Error, throw new 104
événements (fenetreMsg) 202
exception JavaScript 104
extension Mozilla 39

F

fenetreMsg 190
 contenu 193
 gestionnaire d'événement 202
 méthodes et propriétés 198
 modifications pour SOAP 229
feuille de style 30
fichiers .manifest 256
File Manager 6
firefox 1.1 256
firefox-overlay.xul 157
flex (attribut) 36

fonctionnalités de débogage 59
framework Mozilla 2

G

gConfig (variable JavaScript) 116
getCellText() (changements entre
Mozilla 1.7 et 1.8) 264
global.js 122
GUID (Globally Unique Identifier) 243

H

historique
CSS 80
de Mozilla 2
JavaScript 100
PHP 133
SOAP 214

I

id des éléments XUL 57
identification.js 110
identification.xul 28
index-barres-overlay.xul 61
index-forum-overlay.xul 73
index-membres-overlay.xul 75
injections de code 235
install.js 247
install.rdf 243
installed-chrome.txt 43
internationalisation 45

J

JavaScript 99
aide 106
client SOAP 223
déclarer des objets 117
encodage 107
et LDAP 169
exceptions 104
intégration à XUL 105
internationalisation 109
liste des événements 203
syntaxe 101
js.properties 109

L

LDAP (Lightweight Directory Access
Protocol) 169
bind 180
fichiers de configuration 172
file d'attente 179
initialisation 175

listeners 176
opération 180
présentation 171
structure du serveur 172

ldap.js 173
liste des composants XPCOM utilisés 265
liste des membres 75, 182
listitem-iconic 184
locale 48
dossier 40, 47
LXR 54, 153

M

menuitem-iconic 96
mises à jour de XUL Forum 249
Mosaïc 2
-moz-appearance 93
-moz-binding 194
-moz-border-radius 96
-moz-box-orient 90
Mozilla
concept d'habillage 83
et LDAP 170
et le XPFE 4
et XBL 190
framework 2
historique 2
intégration de XUL Forum dans 151
milestones 4
mozilla-overlay.xul 156
-moz-image-region 91
-moz-tree-cell-text 94
-moz-tree-image 95
-moz-tree-row 94
MySQL
pour remplacer LDAP 172
tables de XUL Forum 132
MySQLi 135

N

Necko 10
Netscape 2
sidebars 86
NG Layout 3
NSPR 10

O

onglets 76
open() (méthode JavaScript
incompatible) 264
optimisation
de SOAP, avec WSDL 237

découpage en overlays 56
des sources RDF 142
du premier fichier XUL 32
erreurs RDF 139
répartition du JavaScript sur plusieurs
fichiers 120
XMLHttpRequest asynchrone 116
orient, attribut de <window> 33
overflow (directive CSS) 196
overlay.css 157
overlays 56
distants 60
répartition sur plusieurs fichiers 66
sur des fichiers du navigateur 155

P

palette-overlay.xul 159
params.xml 114
performance, parseur RDF 143
PHP
configuration 214
génération de RDF 132
serveur SOAP 217
piles (FIFO et LIFO) 108
pointeurs de fonction 175
portabilité 12
prototype (modèle objet de JavaScript) 117
proxy XPCOM 179

R

raccourcis clavier dans XUL Forum 165
RDF (Resource Description Framework)
alternative à la connexion LDAP 185
alternatives 143
asynchrone 146
erreurs 139
explications détaillées 126
généré par PHP 132
mise à jour de source 145
rôle dans le XPFE 8
validateur 132
rdf.js 143
RDF/XML 128
rdfmessages.php 133
Roadmap de 1998 4
RSS (RDF Site Summary) 127

S

SeaMonkey
nom de code 4
successeur de Mozilla 4
y installer XUL Forum 261

services web 213
 signer une application 254
 skin (dossier) 40, 83
 SOAP 214
 _appelSoap() (routine JavaScript) 225
 alternatives 215
 communications avec XUL Forum
 (logique) 215
 enregistrement d'un message en
 PHP 221
 identification en JavaScript 227
 identification PHP 219
 initialisation JavaScript 223
 lecture d'un message en PHP 220
 lire un message en JavaScript 227
 mode asynchrone 226
 poster un message en JavaScript 228
 serveur PHP5 217
 vérification de la session en
 JavaScript 224
 SOS fantômes, film 28
 spacers 35
 SpiderMonkey 100
 stringbundle 109, 148
 SVG (Scalable Vector Graphics) 262
 syntaxe
 CSS 273
 DTD 9
 du fichier contents.rdf 41
 JavaScript 101
 PHP 133
 RDF 126
 XML 7

T

tableau (<grid>) 34
 tests unitaires 222
 timestamp 161

U

Unicode 30
 update.rdf 250

V

validateur RDF 132
 visibility (directive CSS) 200

W

window manager 211
 WSDL (Web Services Description
 Language) 237

X

XAML (eXtensible Application Markup
 Language) 12
 XBL (eXtensible Binding Language) 189
 contenu de notre widget 193
 gestionnaire d'événement de notre
 widget 202
 intégration dans XUL Forum 206
 méthodes et propriétés de notre
 widget 198
 modifications à notre widget pour
 SOAP 229
 XML (eXtensible Markup Language)
 B.A.-BA 7
 éditeur de fichiers 28
 espace de nommage 29
 prologue 30
 XMLHttpRequest 114
 asynchrone 116
 XML-RPC 214
 XPCOM (Cross Platform Component
 Object Model) 9
 premier composant 120
 proxy 179
 XPFE (Cross Platform Front-End) 4
 organisation 9
 XPCOM, rôle de 9
 XPIInstall 241
 XUL (XML-based User Interface
 Language)
 avantages 22
 commentaires 56
 DXULi, interpréteur DHTML 24

et CSS 8
 et ECMAScript 8
 et XML 7
 interpréteurs 31
 templates 136
 utilisations 8

XUL Forum
 animation avec JavaScript 99
 arbre des messages 73
 authentification 32
 barre d'outils 69
 découpage 54
 différentes versions 155
 DIT 172
 écran principal 19
 fenêtre principale 53
 fonctionnement 18
 identification (enregistrement des
 choix par défaut) 163
 installation avec XPIInstall 241
 intégration dans Mozilla 151
 intégration du widget XBL 206
 introduction 16
 liste des membres 75
 menus 62
 mises à jour 249
 modèle de boîtes 32
 modules 18
 préférences 161
 raccourcis clavier 165
 services web 213
 tables MySQL 132
 technologies employées 21

XUL Runner 258
 xulforum.css 83
 xulforum.dtd 48
 xulforum.jar 242
 xulforum.manifest 256
 xulforum.xpi 245
 xulforum.xul 43
 xulforum-prefs.js 260

Programmez intelligent avec les Cahiers du Programmeur

XUL

Embarqués dans l'excellent navigateur libre Mozilla Firefox, XUL et le framework XPFE de Mozilla ouvrent la voie à la création et au déploiement d'applications web riches s'appuyant, à la manière d'AJAX, sur les standards du Web (CSS, Javascript, XML, RDF...).

À travers la création d'un forum de discussion, découvrez et maîtrisez la puissance de l'environnement de développement Mozilla, depuis la réalisation d'interfaces graphiques avec XUL et CSS jusqu'à la logique applicative avec Javascript et XPCOM et le déploiement avec XPI. On verra comment intégrer l'application dans le navigateur grâce aux overlays et à RDF et comment recourir à l'API DOM dans JavaScript et PHP 5. L'étude de cas décrit également l'interfaçage avec un annuaire LDAP, la connexion à un serveur SOAP ou la réutilisation de code avec XBL.



Téléchargez l'intégralité du code source XUL et des pages PHP
et retrouvez l'étude de cas en ligne !
www.editions-eyrolles.com

Sommaire

Mozilla, XUL et le XPFE • Organisation du XPFE • L'étude de cas : un forum de discussion • Inscription et identification • Ajout/modification/lecture d'un sujet • Premiers pas en XUL • Contents.rdf et dossier content • Modification du fichier chrome.rdf • Dossier locale • Intégration d'une DTD et début de l'internationalisation • **Fenêtres XUL** • Découpage et séparation en overlays • Barres d'outils, de menu et de statut • Arbre et panneau à onglets • **Rendu CSS** • Mise en forme de texte • Positionnement • CSS spécifique à Mozilla • Couleurs et images • Propriété propre à Mozilla • **Première animation de l'interface avec JavaScript** • Intégration à XUL • Affichage d'erreurs • Multi-langue avec l'objet stringbundle • Manipulations DOM : options à la connexion • Communication avec l'extérieur : récupération d'un fichier de configuration via l'objet XMLHttpRequest • Analyse du fichier avec DOM • Approche des composants XPCOM : fonction include() • **Automatisation avec RDF** • Le format RDF : nœuds, arcs et listes • Sérialisation avec RDF/XML • Génération de RDF avec PHP 5 • Exploitation côté XUL • Amélioration de RDF avec JavaScript : versions synchrone et asynchrone • **Intégration au cœur de Mozilla** • Extension de l'interface du navigateur avec des overlays • Gestion multiple pour Firefox, Thunderbird, Mozilla et la suite • Utiliser les préférences internes de Mozilla • Raccourcis clavier • **Interaction entre JavaScript et LDAP** • Opérations LDAP avec JavaScript et XPCOM • Création d'une file d'attente • Identification avec un simple bind • Recherches sur la liste des membres • **Fonctionnement d'un widget XBL** • Le widget fenetreMsg • Contenu du widget : <content> • Animation avec <implementation> : propriétés, champs, méthodes • Plier, déplier, cacher le fenetreMsg • Widget qui réagit aux événements : <handlers> • **Services web entre client et serveur** • Application de SOAP à XUL Forum • Serveur PHP 5 et client JavaScript • Authentification • Lire et enregistrer un message • Vérifier la session • **Distribution de XUL Forum avec XPI** • Contenu du fichier xulforum.xpi • Création du fichier xulforum.jar • Installation avec install.rdf • Compatibilité avec Mozilla 1.x : install.js • Signaler des mises à jour avec update.rdf • **Annexes** • Vers Firefox 1.5 • XUL Runner • SeaMonkey • SVG, <canvas> • Liste des composants XPCOM • Liste des fichiers de l'application • Rappels CSS • Référence XUL.

Jonathan PROTZENKO

s'est toujours passionné pour les interfaces graphiques, depuis Visual Basic 5 jusqu'à PHP-GTK en passant par le DHTML, Java/AWT et l'API Win32.

Il découvre enfin la solution miracle avec XUL, qu'il s'empresse de faire connaître par des articles et conférences.

Benoît Picaud est consultant senior chez IDEALX, référence française pour la réalisation de logiciels libres. En charge de la conception d'infrastructures de sécurité et de gestion des identités, il s'est intéressé aux problématiques de déploiement des applications en entreprise.

EYROLLES

avaxhome.ws

www.frenchpdf.com